Distributed Document Classifier

1.0

Generated by Doxygen 1.9.8

Chapter 1

Distributed Document Classifier

Author

Wiktor Szewczyk

1.1 Overview

A parallel document classification engine based on GASPI (Global Address Space Programming Interface), inspired by Chapter 9 of *Parallel Programming in C with MPI and OpenMP* by Michael J. Quinn.

This project implements a scalable manager–worker architecture using GASPI's PGAS (Partitioned Global Address Space) model to classify text documents into feature vectors. It uses hashing to match words against a shared dictionary and distributes processing using one-sided communication.

It currently supports only `.txt` documents.

1.2 Architecture

- The **manager process** (rank 0):
  - creates GASPI memory segments for communication (DICT, WORK, RESULT),
  - reads the dictionary file (one keyword per line),
  - writes dictionary to shared DICT_SEGMENT for all workers,
  - scans the input directory for `.txt` files,
  - distributes file paths to workers via WORK_SEGMENT using gaspi_write_notify,
  - receives classified vectors from RESULT_SEGMENT and writes them to the output file.
- Each **worker process** (rank 1 to n-1):
  - reads the dictionary from DICT_SEGMENT,
  - waits for work notifications via gaspi_notify_waitsome,
  - receives file paths from WORK_SEGMENT,
  - tokenizes and lowercases the document content,
  - hashes each word against a fixed-size dictionary hash table,
  - builds a feature vector based on word presence or frequency,
  - writes results to RESULT_SEGMENT using gaspi_write_notify.

1.3 GASPI Memory Segments

The application uses three global memory segments:

- **DICT_SEGMENT_ID (0)**: Shared dictionary data
- **WORK_SEGMENT_ID (1)**: File assignment from manager to workers
- **RESULT_SEGMENT_ID (2)**: Classification results from workers to manager

Each worker has dedicated offsets within segments to avoid memory conflicts.

1.4 Flow Diagram

1.5 Project Structure

```
.
|- include/                        # Header files
|- src/                            # Core runtime: main, manager, worker, utils
|- tests/                          # Criterion unit tests
|- docs/                           # Doxygen config + generated docs
|- make/                           # Makefile submodules (build, test, docs, run)
|- scripts/gaspi_wrapper.sh        # GASPI environment wrapper script
|- scripts/gen_data.py             # Script for generating random data
|- Makefile                        # Entry point Makefile
|- nodes.txt                       # GASPI node configuration file
```

1.6 Requirements

- GCC 12+
- GNU Make
- GASPI/GPI-2 1.5.1+
- MPICH 4.3.0+ (for compilation compatibility)
- CUDA Toolkit 12.1+ (runtime libraries)
- (Optional) `Doxygen 1.9.8+` + `Doxygen-Awesome-CSS`
- (Optional) `Criterion 2.4.1+` for unit testing

1.7 Environment Setup

Before building or running the project, you must set the required environment variables:

```
export GASPI_TARGET_DIR=/path/to/gpi2
export MPICH_TARGET_DIR=/path/to/mpich
export CUDA_TARGET_DIR=/path/to/cuda
```

1.8 Build

```
make build
```

Builds the executable at:

```
./build/bin/ddc
```

1.9 Running the Classifier

```
make run GASPI_FLAGS='-n 8' RUN_FLAGS='input/ dict.txt out.txt'
```

- `GASPI_FLAGS`: passed to `gaspi_run` (default: `-n 16`)
- `RUN_FLAGS`: arguments for `ddc` (default: `./example/input/ ./example/dict.txt ./example/output/res`

1.10 Running Tests

```
make test
```

Runs all unit tests with Criterion.

1.11 Documentation

```
git submodule update --init --update
make docs
```

- HTML docs: `docs/html/index.html`

1.12 License

MIT License. See `LICENSE` for details.

1.13 References

- Michael J. Quinn, *Parallel Programming in C with MPI and OpenMP*, Chapter 9
- GASPI/GPI-2
- Criterion
- Doxygen
- Doxygen-Awesome-CSS

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

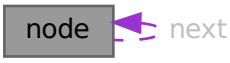
Chapter 4

Data Structure Documentation

4.1 node Struct Reference

`#include <hash_table.h>`

Collaboration diagram for node:



Data Fields

- int index
  Position in feature vector.
- struct node * next
  Next node in chain.
- char * word
  Keyword string.

4.1.1 Detailed Description

Definition at line 23 of file hash_table.h.

4.1.2 Field Documentation

4.1.2.1 index

`int node::index`

Position in feature vector.

Definition at line 26 of file hash_table.h.

4.1.2.2 next

```
struct node* node::next
```

Next node in chain.

Definition at line 27 of file hash_table.h.

4.1.2.3 word

```
char* node::word
```

Keyword string.

Definition at line 25 of file hash_table.h.

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

- hash_table.h

4.2 Node Struct Reference

Represents a dictionary word in a linked list at a hash slot.

```
#include <hash_table.h>
```

4.2.1 Detailed Description

Represents a dictionary word in a linked list at a hash slot.

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

- hash_table.h

Chapter 5

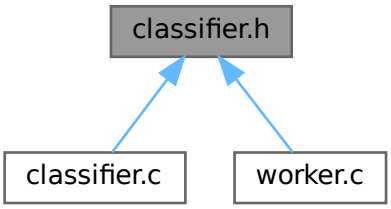
File Documentation

5.1 mainpage.dox File Reference

5.2 classifier.h File Reference

Tokenization and text classification logic.

This graph shows which files directly or indirectly include this file:



Macros

- #define CLASSIFY_TOKENS " \n\t.,;:!?()[]{}\"'-"

  Token delimiters for splitting text into words.

Functions

- void classify_text (const char *text, int result[], int keyword_count)

  Generates a classification vector from input text.

5.2.1 Detailed Description

Tokenization and text classification logic.

Author

Wiktor Szewczyk

Definition in file classifier.h.

5.2.2 Macro Definition Documentation

5.2.2.1 CLASSIFY_TOKENS

```
#define CLASSIFY_TOKENS " \n\t.,;:!?()[]{}\"'-"
```

Token delimiters for splitting text into words.

Definition at line 11 of file classifier.h.

5.2.3 Function Documentation

5.2.3.1 classify_text()

```
void classify_text (
            const char * text,
            int result[],
            int keyword_count )
```

Generates a classification vector from input text.

The function tokenizes the input string and compares each token to a global hash table containing dictionary keywords.

Parameters

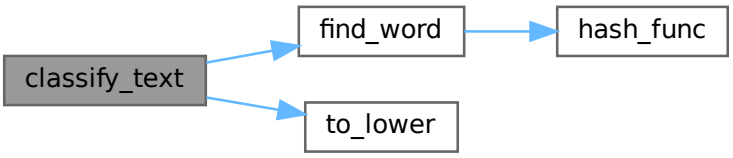
text	Input document text
result	Output vector (must be zeroed and sized to keyword_count)
keyword_count	Number of keywords in dictionary

Definition at line 13 of file classifier.c.

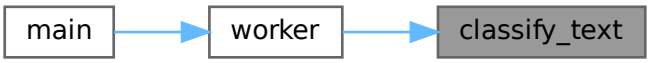
References CLASSIFY_TOKENS, find_word(), MAX_DOC_SIZE, and to_lower().

Referenced by worker().

Here is the call graph for this function:



Here is the caller graph for this function:



5.3 classifier.h

Go to the documentation of this file.

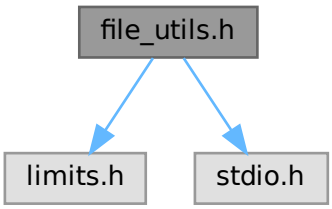
```
00001
00007 #ifndef CLASSIFIER_H
00008 #define CLASSIFIER_H
00009
00011 #define CLASSIFY_TOKENS " \n\t.,;:!?()[]{}\"'-"
00012
00023 void classify_text(const char *text, int result[], int keyword_count);
00024
00025 #endif
```

5.4 file_utils.h File Reference

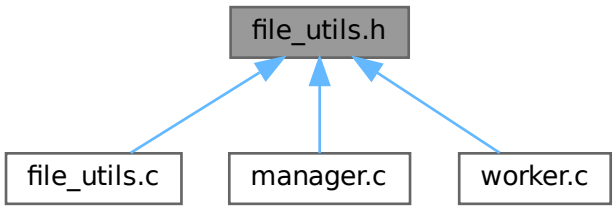
Utility functions for files operations.

```
#include <limits.h>
#include <stdio.h>
```

Include dependency graph for file_utils.h:



This graph shows which files directly or indirectly include this file:



Functions

- void broadcast_dictionary (char ∗keywords[ ], int num_keywords)
  Broadcasts dictionary from manager to workers.
- int list_txt_files (const char ∗dir_path, char files[ ][PATH_MAX], int max_files)
  Lists all .txt files in a directory.
- FILE ∗ open_output_file (const char ∗output_path)
  Opens the result output file for writing.
- int read_dictionary (const char ∗dict_path, char ∗keywords[ ], int max_keywords)
  Reads dictionary and builds global keyword list and hash table.
- int read_file_content (const char ∗filename, char ∗buffer, size_t bufsize)
  Reads entire content of a file into buffer.
- void receive_dictionary (char ∗keywords[ ], int ∗num_keywords)
  Receives dictionary in a worker and builds local keyword array and hash table.

5.4.1 Detailed Description

Utility functions for files operations.

Author

Wiktor Szewczyk

Definition in file file_utils.h.

5.4.2 Function Documentation

5.4.2.1 broadcast_dictionary()

```
void broadcast_dictionary (
            char * keywords[],
            int num_keywords )
```

Broadcasts dictionary from manager to workers.

Parameters

keywords	Array of keyword strings
num_keywords	Number of keywords

Definition at line 84 of file file_utils.c.

References DICT_SEGMENT_ID, GASPI_CHECK, and MAX_WORD_LEN.

Referenced by manager().

Here is the caller graph for this function:

5.4.2.2 list_txt_files()

```
int list_txt_files (
            const char * dir_path,
            char files[][PATH_MAX],
            int max_files )
```

Lists all .txt files in a directory.

Parameters

dir_path	Path to the directory
files	Output array of full file paths
max_files	Maximum number of files to find

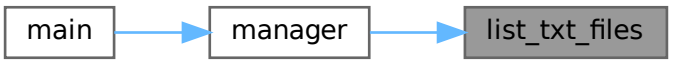
Returns

Number of files found, or -1 on error

Definition at line 19 of file file_utils.c.

Referenced by manager().

Here is the caller graph for this function:



5.4.2.3 open_output_file()

```
FILE * open_output_file (
            const char * output_path )
```

Opens the result output file for writing.

Parameters

output_path	Output file path

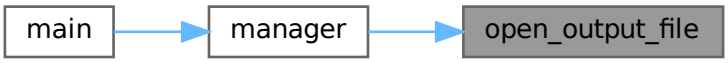
Returns

FILE* handle or NULL on failure

Definition at line 53 of file file_utils.c.

Referenced by manager().

Here is the caller graph for this function:



5.4.2.4 read_dictionary()

```
int read_dictionary (
            const char * dict_path,
            char * keywords[],
            int max_keywords )
```

Reads dictionary and builds global keyword list and hash table.

Parameters

dict_path	Path to dictionary file
keywords	Output array of allocated keyword strings
max_keywords	Maximum allowed keyword count

Returns

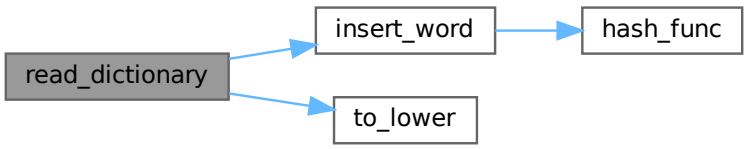
Number of keywords read, or -1 on error

Definition at line 58 of file file_utils.c.

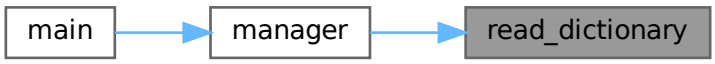
References insert_word(), MAX_WORD_LEN, and to_lower().

Referenced by manager().

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.5 read_file_content()

```
int read_file_content (
            const char * filename,
            char * buffer,
            size_t bufsize )
```

Reads entire content of a file into buffer.

Parameters

filename	File to read
buffer	Destination buffer
bufsize	Maximum size of buffer

Returns

0 on success, -1 on error

Definition at line 41 of file file_utils.c.

Referenced by worker().

Here is the caller graph for this function:

5.4.2.6 receive_dictionary()

```
void receive_dictionary (
            char * keywords[],
            int * num_keywords )
```

Receives dictionary in a worker and builds local keyword array and hash table.

Parameters

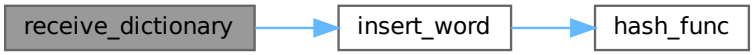
keywords	Output array of keyword strings
num_keywords	Output number of keywords

Definition at line 113 of file file_utils.c.

References DICT_SEGMENT_ID, GASPI_CHECK, insert_word(), and MAX_WORD_LEN.

Referenced by worker().

Here is the call graph for this function:



Here is the caller graph for this function:

5.5 file_utils.h

Go to the documentation of this file.

```
00001
00007 #ifndef FILE_UTILS_H
00008 #define FILE_UTILS_H
00009
00010 #include <limits.h>
00011 #include <stdio.h>
00012
00021 int list_txt_files(const char *dir_path, char files[][PATH_MAX], int max_files);
00022
00031 int read_file_content(const char *filename, char *buffer, size_t bufsize);
00032
00039 FILE *open_output_file(const char *output_path);
00040
00049 int read_dictionary(const char *dict_path, char *keywords[], int max_keywords);
00050
00057 void broadcast_dictionary(char *keywords[], int num_keywords);
00058
00065 void receive_dictionary(char *keywords[], int *num_keywords);
00066
00067 #endif
```

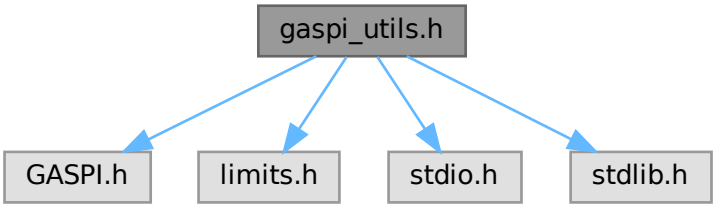
5.6 gaspi_utils.h File Reference

Utility macros and functions for GASPI error handling.

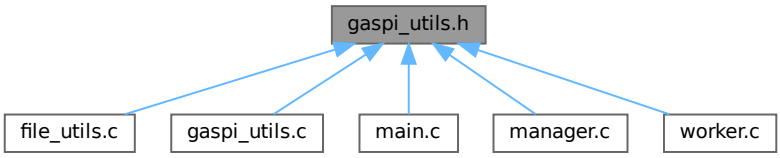
```
#include <GASPI.h>
#include <limits.h>
#include <stdio.h>
```

```
#include <stdlib.h>
```

Include dependency graph for gaspi_utils.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define DICT_SEGMENT_ID 0
- #define GASPI_CHECK(call)

  Check if GASPI call returned success, exit on error.
- #define RESULT_OFFSET(rank) ((rank - 1) ∗ WORKER_DATA_SIZE)
- #define RESULT_SEGMENT_ID 2
- #define WORK_OFFSET(rank) ((rank - 1) ∗ PATH_MAX)
- #define WORK_SEGMENT_ID 1
- #define WORKER_DATA_SIZE (PATH_MAX + MAX_KEYWORDS ∗ sizeof(int))

Functions

- void cleanup_gaspi_segments (void)

  Cleans up GASPI segments.
- void init_gaspi_segments (void)

  Initializes GASPI segments for dictionary and work data.

5.6.1 Detailed Description

Utility macros and functions for GASPI error handling.

Author

Wiktor Szewczyk

Definition in file gaspi_utils.h.

5.6.2 Macro Definition Documentation

5.6.2.1 DICT_SEGMENT_ID

```
#define DICT_SEGMENT_ID 0
```

Definition at line 33 of file gaspi_utils.h.

5.6.2.2 GASPI_CHECK

```
#define GASPI_CHECK(
              call )
```

Value:

```
    do                                                                          \
    {                                                                           \
        gaspi_return_t ret = (call);                                            \
        if (ret != GASPI_SUCCESS)                                               \
        {                                                                       \
            fprintf(stderr, "GASPI error at %s:%d: %s returned %d\n", __FILE__, \
                    __LINE__, #call, ret);                                      \
            gaspi_proc_term(GASPI_BLOCK);                                       \
            exit(EXIT_FAILURE);                                                 \
        }                                                                       \
    } while (0)
```

Check if GASPI call returned success, exit on error.

Parameters

call	The GASPI function call to check

Definition at line 19 of file gaspi_utils.h.

5.6.2.3 RESULT_OFFSET

```
#define RESULT_OFFSET(
              rank ) ((rank - 1) * WORKER_DATA_SIZE)
```

Definition at line 43 of file gaspi_utils.h.

5.6.2.4 RESULT_SEGMENT_ID

```
#define RESULT_SEGMENT_ID 2
```

Definition at line 35 of file gaspi_utils.h.

5.6.2.5 WORK_OFFSET

```
#define WORK_OFFSET(
              rank ) ((rank - 1) * PATH_MAX)
```

Definition at line 41 of file gaspi_utils.h.

5.6.2.6 WORK_SEGMENT_ID

```
#define WORK_SEGMENT_ID 1
```

Definition at line 34 of file gaspi_utils.h.

5.6.2.7 WORKER_DATA_SIZE

```
#define WORKER_DATA_SIZE (PATH_MAX + MAX_KEYWORDS * sizeof(int))
```

Definition at line 38 of file gaspi_utils.h.

5.6.3 Function Documentation

5.6.3.1 cleanup_gaspi_segments()

```
void cleanup_gaspi_segments (
             void  )
```

Cleans up GASPI segments.

Definition at line 24 of file gaspi_utils.c.

References DICT_SEGMENT_ID, and WORK_SEGMENT_ID.

5.6.3.2 init_gaspi_segments()

```
void init_gaspi_segments (
             void  )
```

Initializes GASPI segments for dictionary and work data.

Definition at line 10 of file gaspi_utils.c.

References DICT_SEGMENT_ID, GASPI_CHECK, MAX_DOC_SIZE, MAX_KEYWORDS, MAX_WORD_LEN, and WORK_SEGMENT_ID.

5.7 gaspi_utils.h

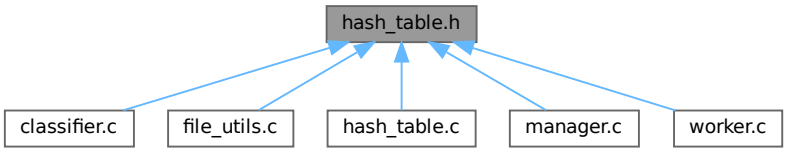
Go to the documentation of this file.

```
00001
00007 #ifndef GASPI_UTILS_H
00008 #define GASPI_UTILS_H
00009
00010 #include <GASPI.h>
00011 #include <limits.h>
00012 #include <stdio.h>
00013 #include <stdlib.h>
00014
00019 #define GASPI_CHECK(call)                                                      \
00020     do                                                                         \
00021     {                                                                          \
00022         gaspi_return_t ret = (call);                                           \
00023         if (ret != GASPI_SUCCESS)                                              \
00024         {                                                                      \
00025             fprintf(stderr, "GASPI error at %s:%d: %s returned %d\n", __FILE__,  \
00026                     __LINE__, #call, ret);                                     \
00027             gaspi_proc_term(GASPI_BLOCK);                                      \
00028             exit(EXIT_FAILURE);                                                \
00029         }                                                                      \
00030     } while (0)
00031
00032 // Constants for GASPI segments
00033 #define DICT_SEGMENT_ID 0
00034 #define WORK_SEGMENT_ID 1
00035 #define RESULT_SEGMENT_ID 2
00036
00037 // Per worker segment size
00038 #define WORKER_DATA_SIZE (PATH_MAX + MAX_KEYWORDS * sizeof(int))
00039
00040 // Per worker work segment offset
00041 #define WORK_OFFSET(rank) ((rank - 1) * PATH_MAX)
00042 // Per worker result segment offset
00043 #define RESULT_OFFSET(rank) ((rank - 1) * WORKER_DATA_SIZE)
00044
00048 void init_gaspi_segments(void);
00049
00053 void cleanup_gaspi_segments(void);
00054
00055 #endif /* GASPI_UTILS_H */
```

5.8 hash_table.h File Reference

Simple fixed-size chained hash table for word lookup.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct node

Macros

- #define HASH_SIZE 101

  Size of the hash table (number of buckets).

Typedefs

- typedef struct node Node

Functions

- void clear_hash_table (void)

  Frees all allocated hash table entries.
- int find_word (const char ∗word)

  Finds a word in the hash table.
- int hash_func (const char ∗s)

  Hash function for a word.
- void insert_word (const char ∗word, int index)

  Inserts a keyword into the hash table.
- void to_lower (char ∗str)

  Converts a string to lowercase in-place.

Variables

- Node ∗ hash_table [101]

  Global hash table structure.

5.8.1 Detailed Description

Simple fixed-size chained hash table for word lookup.

Author

Wiktor Szewczyk

Definition in file hash_table.h.

5.8.2 Macro Definition Documentation

5.8.2.1 HASH_SIZE

```
#define HASH_SIZE 101
```

Size of the hash table (number of buckets).

The hash table uses fixed-size separate chaining with linked lists. A prime number is chosen to reduce the likelihood of collisions and distribute keys uniformly.

Definition at line 17 of file hash_table.h.

5.8.3 Typedef Documentation

5.8.3.1 Node

```
typedef struct node Node
```

5.8.4 Function Documentation

5.8.4.1 clear_hash_table()

```
void clear_hash_table (
            void  )
```

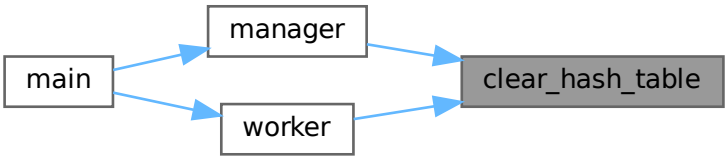
Frees all allocated hash table entries.

Definition at line 48 of file hash_table.c.

References HASH_SIZE, and hash_table.

Referenced by manager(), and worker().

Here is the caller graph for this function:



5.8.4.2 find_word()

```
int find_word (
            const char * word )
```

Finds a word in the hash table.

Parameters

word	Word to find

Returns

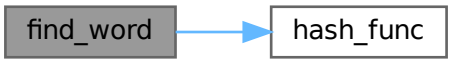
Index in dictionary, or -1 if not found

Definition at line 35 of file hash_table.c.

References hash_func(), and hash_table.

Referenced by classify_text().

Here is the call graph for this function:



Here is the caller graph for this function:

5.8.4.3 hash_func()

```
int hash_func (
            const char * s )
```

Hash function for a word.

Parameters

s	Word to hash

Returns

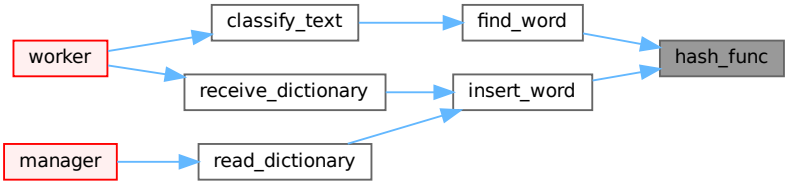
Index in hash table

Definition at line 15 of file hash_table.c.

References HASH_SIZE.

Referenced by find_word(), and insert_word().

Here is the caller graph for this function:



5.8.4.4 insert_word()

```
void insert_word (
            const char * word,
            int index )
```

Inserts a keyword into the hash table.

Parameters

word	Keyword string
index	Index in the dictionary

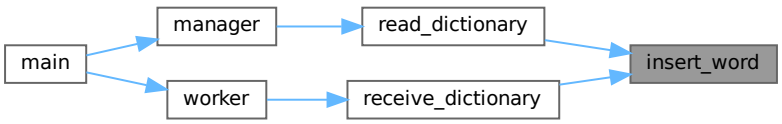
Definition at line 25 of file hash_table.c.

References hash_func(), and hash_table.

Referenced by read_dictionary(), and receive_dictionary().

Here is the call graph for this function:

Here is the caller graph for this function:



5.8.4.5 to_lower()

```
void to_lower (
            char * str )
```

Converts a string to lowercase in-place.

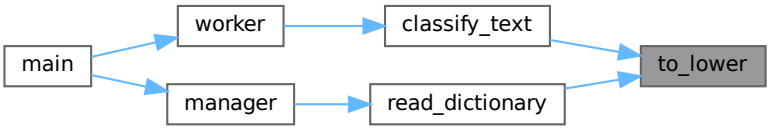
Parameters

str	String to lowercase

Definition at line 64 of file hash_table.c.

Referenced by classify_text(), and read_dictionary().

Here is the caller graph for this function:



5.8.5 Variable Documentation

5.8.5.1 hash_table

```
Node* hash_table[101]  [extern]
```

Global hash table structure.

Definition at line 13 of file hash_table.c.

Referenced by clear_hash_table(), find_word(), and insert_word().

5.9 hash_table.h

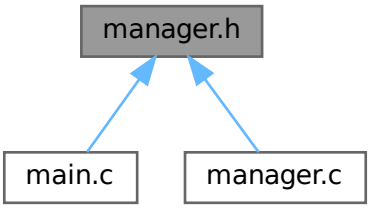
Go to the documentation of this file.

```
00001
00007 #ifndef HASH_TABLE_H
00008 #define HASH_TABLE_H
00009
00017 #define HASH_SIZE 101
00018
00023 typedef struct node
00024 {
00025     char *word;
00026     int index;
00027     struct node *next;
00028 } Node;
00029
00031 extern Node *hash_table[HASH_SIZE];
00032
00039 int hash_func(const char *s);
00040
00047 void insert_word(const char *word, int index);
00048
00055 int find_word(const char *word);
00056
00060 void clear_hash_table(void);
00061
00067 void to_lower(char *str);
00068
00069 #endif
```

5.10 manager.h File Reference

Interface for the manager process.

This graph shows which files directly or indirectly include this file:



Functions

- void manager (const char *input_dir, const char *dict_file, const char *output_file, int processes_num)
  Entry point for the manager process.

5.10.1 Detailed Description

Interface for the manager process.

Author

Wiktor Szewczyk

Definition in file manager.h.

5.10.2 Function Documentation

5.10.2.1 manager()

```
void manager (
            const char * input_dir,
            const char * dict_file,
            const char * output_file,
            int processes_num )
```

Entry point for the manager process.

The manager is responsible for:

- Reading and broadcasting the dictionary
- Scanning the input directory for text files
- Distributing work to worker processes
- Collecting and writing classification results

Parameters

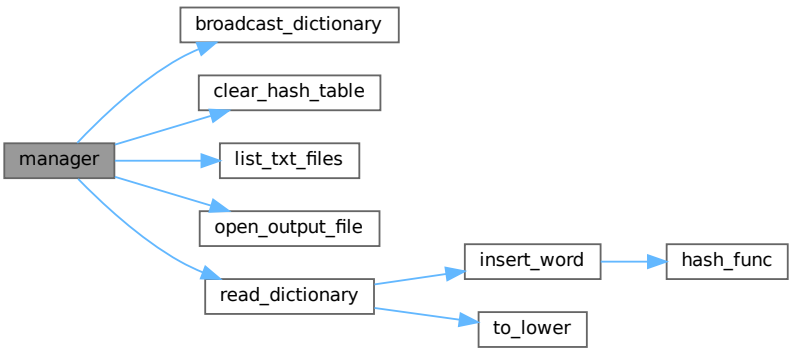
input_dir	Path to directory containing .txt files
dict_file	Path to the dictionary file
output_file	Path where results will be written
processes_num	Number of GASPI processes

Definition at line 20 of file manager.c.

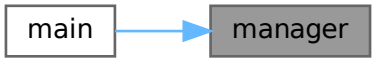
References broadcast_dictionary(), clear_hash_table(), DICT_SEGMENT_ID, GASPI_CHECK, list_txt_files(), MAX_FILES, MAX_KEYWORDS, MAX_WORD_LEN, open_output_file(), read_dictionary(), RESULT_OFFSET, RESULT_SEGMENT_ID, SHUTDOWN_NOTIF, WORK_AVAILABLE_NOTIF, WORK_DONE_NOTIF, WORK_NOTIF_ID, WORK_OFFSET, WORK_SEGMENT_ID, and WORKER_DATA_SIZE.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.11 manager.h

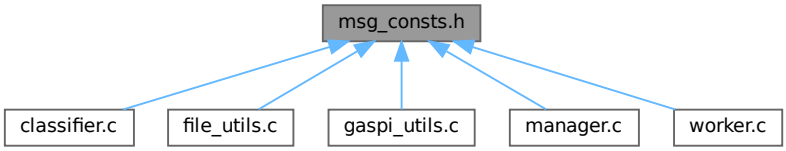
Go to the documentation of this file.

```
00001
00007 #ifndef MANAGER_H
00008 #define MANAGER_H
00009
00024 void manager(const char *input_dir, const char *dict_file, const char *output_file,
00025              int processes_num);
00026
00027 #endif
```

5.12 msg_consts.h File Reference

Constants for GASPI messaging and buffer sizes.

This graph shows which files directly or indirectly include this file:



Macros

- #define MAX_DOC_SIZE 4096

  Maximum size of a document in bytes.
- #define MAX_FILES 1024

  Maximum number of files.
- #define MAX_KEYWORDS 256

  Maximum number of dictionary keywords.
- #define MAX_WORD_LEN 64

  Maximum length of a single keyword.
- #define RESULT_NOTIF_ID 1

Result notification ID.

- #define SHUTDOWN_NOTIF 0

  GASPI notification values.
- #define WORK_AVAILABLE_NOTIF 1

  Work available notification.
- #define WORK_DONE_NOTIF 2

  Work completed notification.
- #define WORK_NOTIF_ID 0

  GASPI notification IDs.

5.12.1 Detailed Description

Constants for GASPI messaging and buffer sizes.

Author

Wiktor Szewczyk

Definition in file msg_consts.h.

5.12.2 Macro Definition Documentation

5.12.2.1 MAX_DOC_SIZE

```
#define MAX_DOC_SIZE 4096
```

Maximum size of a document in bytes.

Definition at line 11 of file msg_consts.h.

5.12.2.2 MAX_FILES

```
#define MAX_FILES 1024
```

Maximum number of files.

Definition at line 20 of file msg_consts.h.

5.12.2.3 MAX_KEYWORDS

```
#define MAX_KEYWORDS 256
```

Maximum number of dictionary keywords.

Definition at line 14 of file msg_consts.h.

5.12.2.4 MAX_WORD_LEN

```
#define MAX_WORD_LEN 64
```

Maximum length of a single keyword.

Definition at line 17 of file msg_consts.h.

5.12.2.5 RESULT_NOTIF_ID

```
#define RESULT_NOTIF_ID 1
```

Result notification ID.

Definition at line 29 of file msg_consts.h.

5.12.2.6 SHUTDOWN_NOTIF

```
#define SHUTDOWN_NOTIF 0
```

GASPI notification values.

Shutdown signal

Definition at line 23 of file msg_consts.h.

5.12.2.7 WORK_AVAILABLE_NOTIF

```
#define WORK_AVAILABLE_NOTIF 1
```

Work available notification.

Definition at line 24 of file msg_consts.h.

5.12.2.8 WORK_DONE_NOTIF

```
#define WORK_DONE_NOTIF 2
```

Work completed notification.

Definition at line 25 of file msg_consts.h.

5.12.2.9 WORK_NOTIF_ID

```
#define WORK_NOTIF_ID 0
```

GASPI notification IDs.

Work notification ID

Definition at line 28 of file msg_consts.h.

5.13 msg_consts.h

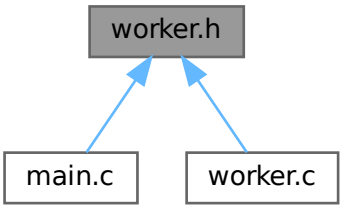
Go to the documentation of this file.

```
00001
00007 #ifndef MSG_CONSTS_H
00008 #define MSG_CONSTS_H
00009
00011 #define MAX_DOC_SIZE 4096
00012
00014 #define MAX_KEYWORDS 256
00015
00017 #define MAX_WORD_LEN 64
00018
00020 #define MAX_FILES 1024
00021
00023 #define SHUTDOWN_NOTIF 0
00024 #define WORK_AVAILABLE_NOTIF 1
00025 #define WORK_DONE_NOTIF 2
00026
00028 #define WORK_NOTIF_ID 0
00029 #define RESULT_NOTIF_ID 1
00030
00031 #endif
```

5.14 worker.h File Reference

Interface for worker process.

This graph shows which files directly or indirectly include this file:



Functions

- void worker (void)

  Entry point for each GASPI worker process.

5.14.1 Detailed Description

Interface for worker process.

Author

Wiktor Szewczyk

Definition in file worker.h.

5.14.2 Function Documentation

5.14.2.1 worker()

```
void worker (
            void  )
```

Entry point for each GASPI worker process.

Each worker:

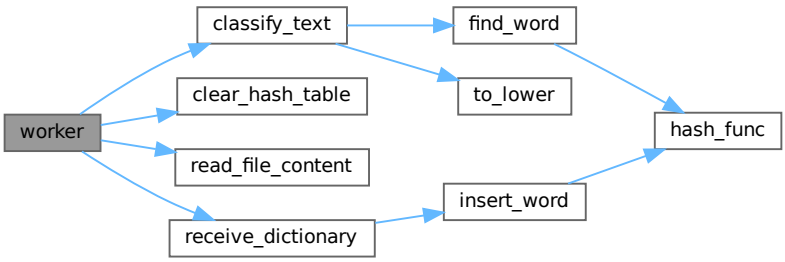
- Receives the dictionary from the manager
- Processes assigned documents
- Sends back feature vectors to the manager

Definition at line 21 of file worker.c.

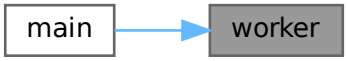
References classify_text(), clear_hash_table(), GASPI_CHECK, MAX_DOC_SIZE, MAX_KEYWORDS, read_file_content(), receive_dictionary(), RESULT_OFFSET, RESULT_SEGMENT_ID, SHUTDOWN_NOTIF, WORK_AVAILABLE_NOTIF, WORK_DONE_NOTIF, WORK_NOTIF_ID, WORK_OFFSET, and WORK_SEGMENT_ID.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.15 worker.h

Go to the documentation of this file.

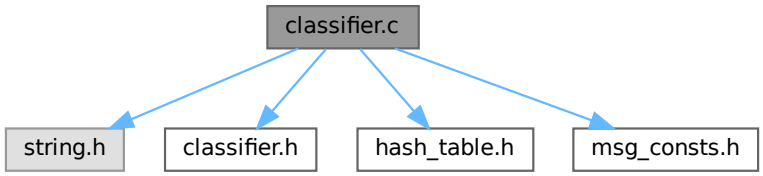
```
00001
00007 #ifndef WORKER_H
00008 #define WORKER_H
00009
00018 void worker(void);
00019
00020 #endif
```

5.16 classifier.c File Reference

Implements document classification using tokenization and a hash table.

```
#include <string.h>
#include "classifier.h"
#include "hash_table.h"
#include "msg_consts.h"
```

Include dependency graph for classifier.c:



Functions

- void classify_text (const char *text, int result[ ], int keyword_count)

  Generates a classification vector from input text.

5.16.1 Detailed Description

Implements document classification using tokenization and a hash table.

Author

Wiktor Szewczyk

Definition in file classifier.c.

5.16.2 Function Documentation

5.16.2.1 classify_text()

```
void classify_text (
            const char * text,
            int result[],
            int keyword_count )
```

Generates a classification vector from input text.

The function tokenizes the input string and compares each token to a global hash table containing dictionary keywords.

Parameters

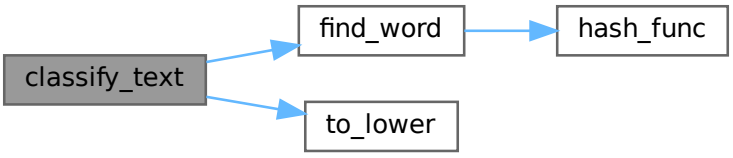
text	Input document text
result	Output vector (must be zeroed and sized to keyword_count)
keyword_count	Number of keywords in dictionary

Definition at line 13 of file classifier.c.

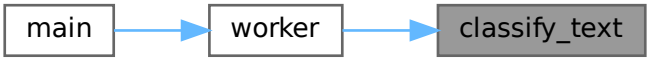
References CLASSIFY_TOKENS, find_word(), MAX_DOC_SIZE, and to_lower().

Referenced by worker().

Here is the call graph for this function:



Here is the caller graph for this function:



5.17 classifier.c

Go to the documentation of this file.

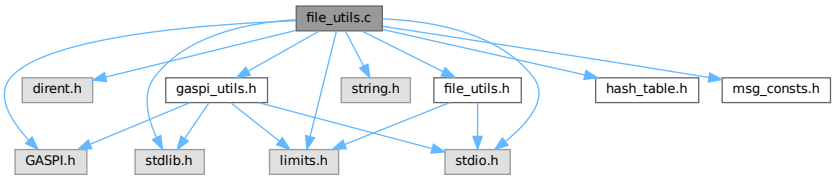
```
00001
00007 #include <string.h>
00008
00009 #include "classifier.h"
00010 #include "hash_table.h"
00011 #include "msg_consts.h"
00012
00013 void classify_text(const char *text, int result[], int keyword_count)
00014 {
00015     memset(result, 0, sizeof(int) * keyword_count);
00016
00017     char buffer[MAX_DOC_SIZE];
00018     strncpy(buffer, text, sizeof(buffer));
00019     buffer[sizeof(buffer) - 1] = '\0';
00020
00021     char *token = strtok(buffer, CLASSIFY_TOKENS);
00022     while (token)
00023     {
00024         to_lower(token);
00025         int index = find_word(token);
00026         if (index != -1)
00027             result[index]++;
00028         token = strtok(NULL, CLASSIFY_TOKENS);
00029     }
00030 }
```

5.18 file_utils.c File Reference

File reading, dictionary parsing, and dictionary broadcasting over GASPI.

```
#include <GASPI.h>
#include <dirent.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "file_utils.h"
#include "gaspi_utils.h"
#include "hash_table.h"
#include "msg_consts.h"
```

Include dependency graph for file_utils.c:



Functions

- void broadcast_dictionary (char ∗keywords[ ], int num_keywords)

  Broadcasts dictionary from manager to workers.
- int list_txt_files (const char ∗dir_path, char files[ ][PATH_MAX], int max_files)

  Lists all .txt files in a directory.

- FILE * open_output_file (const char *output_path)

  Opens the result output file for writing.
- int read_dictionary (const char *dict_path, char *keywords[ ], int max_keywords)

  Reads dictionary and builds global keyword list and hash table.
- int read_file_content (const char *filename, char *buffer, size_t bufsize)

  Reads entire content of a file into buffer.
- void receive_dictionary (char *keywords[ ], int *num_keywords)

  Receives dictionary in a worker and builds local keyword array and hash table.

5.18.1 Detailed Description

File reading, dictionary parsing, and dictionary broadcasting over GASPI.

Author

Wiktor Szewczyk

Definition in file file_utils.c.

5.18.2 Function Documentation

5.18.2.1 broadcast_dictionary()

```
void broadcast_dictionary (
            char * keywords[],
            int num_keywords )
```

Broadcasts dictionary from manager to workers.

Parameters

keywords	Array of keyword strings
num_keywords	Number of keywords

Definition at line 84 of file file_utils.c.

References DICT_SEGMENT_ID, GASPI_CHECK, and MAX_WORD_LEN.

Referenced by manager().

Here is the caller graph for this function:

5.18.2.2 list_txt_files()

```
int list_txt_files (
            const char * dir_path,
            char files[][PATH_MAX],
            int max_files )
```

Lists all .txt files in a directory.

Parameters

dir_path	Path to the directory
files	Output array of full file paths
max_files	Maximum number of files to find

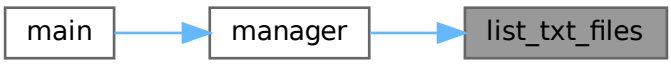
Returns

Number of files found, or -1 on error

Definition at line 19 of file file_utils.c.

Referenced by manager().

Here is the caller graph for this function:



5.18.2.3 open_output_file()

```
FILE * open_output_file (
            const char * output_path )
```

Opens the result output file for writing.

Parameters

output_path	Output file path

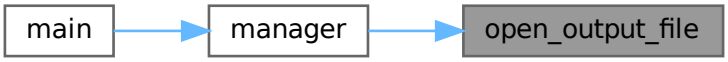
Returns

FILE∗ handle or NULL on failure

Definition at line 53 of file file_utils.c.

Referenced by manager().

Here is the caller graph for this function:



5.18.2.4 read_dictionary()

```
int read_dictionary (
            const char * dict_path,
            char * keywords[],
            int max_keywords )
```

Reads dictionary and builds global keyword list and hash table.

Parameters

dict_path	Path to dictionary file
keywords	Output array of allocated keyword strings
max_keywords	Maximum allowed keyword count

Returns

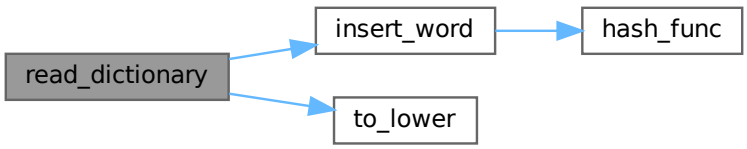
Number of keywords read, or -1 on error

Definition at line 58 of file file_utils.c.

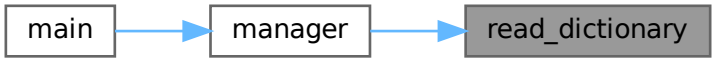
References insert_word(), MAX_WORD_LEN, and to_lower().

Referenced by manager().

Here is the call graph for this function:



Here is the caller graph for this function:



5.18.2.5 read_file_content()

```
int read_file_content (
            const char * filename,
            char * buffer,
            size_t bufsize )
```

Reads entire content of a file into buffer.

Parameters

filename	File to read
buffer	Destination buffer
bufsize	Maximum size of buffer

Returns

0 on success, -1 on error

Definition at line 41 of file file_utils.c.

Referenced by worker().

Here is the caller graph for this function:

5.18.2.6 receive_dictionary()

```
void receive_dictionary (
            char * keywords[],
            int * num_keywords )
```

Receives dictionary in a worker and builds local keyword array and hash table.

Parameters

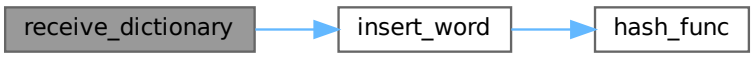
keywords	Output array of keyword strings
num_keywords	Output number of keywords

Definition at line 113 of file file_utils.c.

References DICT_SEGMENT_ID, GASPI_CHECK, insert_word(), and MAX_WORD_LEN.

Referenced by worker().

Here is the call graph for this function:



Here is the caller graph for this function:

5.19 file_utils.c

Go to the documentation of this file.

```
00001
00007 #include <GASPI.h>
00008 #include <dirent.h>
00009 #include <limits.h>
00010 #include <stdio.h>
00011 #include <stdlib.h>
00012 #include <string.h>
00013
00014 #include "file_utils.h"
00015 #include "gaspi_utils.h"
00016 #include "hash_table.h"
00017 #include "msg_consts.h"
00018
00019 int list_txt_files(const char *dir_path, char files[][PATH_MAX], int max_files)
00020 {
00021     DIR *dir = opendir(dir_path);
00022     if (!dir)
00023         return -1;
00024
00025     struct dirent *entry;
00026     int count = 0;
00027
00028     while ((entry = readdir(dir)) != NULL && count < max_files)
00029     {
```

```
00030         if (strstr(entry->d_name, ".txt"))
00031         {
00032             snprintf(files[count], PATH_MAX, "%s/%s", dir_path, entry->d_name);
00033             count++;
00034         }
00035     }
00036
00037     closedir(dir);
00038     return count;
00039 }
00040
00041 int read_file_content(const char *filename, char *buffer, size_t bufsize)
00042 {
00043     FILE *f = fopen(filename, "r");
00044     if (!f)
00045         return -1;
00046
00047     size_t n = fread(buffer, 1, bufsize - 1, f);
00048     buffer[n] = '\0';
00049     fclose(f);
00050     return 0;
00051 }
00052
00053 FILE *open_output_file(const char *output_path)
00054 {
00055     return fopen(output_path, "w");
00056 }
00057
00058 int read_dictionary(const char *dict_path, char *keywords[], int max_keywords)
00059 {
00060     FILE *df = fopen(dict_path, "r");
00061     if (!df)
00062         return -1;
00063
00064     int count = 0;
00065     char word[MAX_WORD_LEN] = {0};
00066
00067     while (fscanf(df, "%63s", word) == 1 && count < max_keywords)
00068     {
00069         to_lower(word);
00070         keywords[count] = strdup(word);
00071         if (!keywords[count])
00072         {
00073             fclose(df);
00074             return -2;
00075         }
00076         insert_word(word, count);
00077         count++;
00078     }
00079
00080     fclose(df);
00081     return count;
00082 }
00083
00084 void broadcast_dictionary(char *keywords[], int num_keywords)
00085 {
00086     gaspi_rank_t size;
00087     GASPI_CHECK(gaspi_proc_num(&size));
00088
00089     gaspi_pointer_t dict_seg_ptr;
00090     GASPI_CHECK(gaspi_segment_ptr(DICT_SEGMENT_ID, &dict_seg_ptr));
00091
00092     // Zapisz liczbę słów kluczowych
00093     memcpy(dict_seg_ptr, &num_keywords, sizeof(int));
00094
00095     // Zapisz słowa kluczowe
00096     for (int i = 0; i < num_keywords; i++)
00097     {
00098         strncpy((char *) dict_seg_ptr + sizeof(int) + i * MAX_WORD_LEN, keywords[i],
00099                 MAX_WORD_LEN - 1);
00100     }
00101
00102     // Wyślij do wszystkich workerów
00103     for (gaspi_rank_t rank = 1; rank < size; rank++)
00104     {
00105         GASPI_CHECK(gaspi_write(DICT_SEGMENT_ID, 0, rank, DICT_SEGMENT_ID, 0,
00106                                 sizeof(int) + num_keywords * MAX_WORD_LEN, 0,
00107                                 GASPI_BLOCK));
00108     }
00109
00110     GASPI_CHECK(gaspi_wait(0, GASPI_BLOCK));
00111 }
00112
00113 void receive_dictionary(char *keywords[], int *num_keywords)
00114 {
00115     gaspi_pointer_t dict_seg_ptr;
00116     GASPI_CHECK(gaspi_segment_ptr(DICT_SEGMENT_ID, &dict_seg_ptr));
```

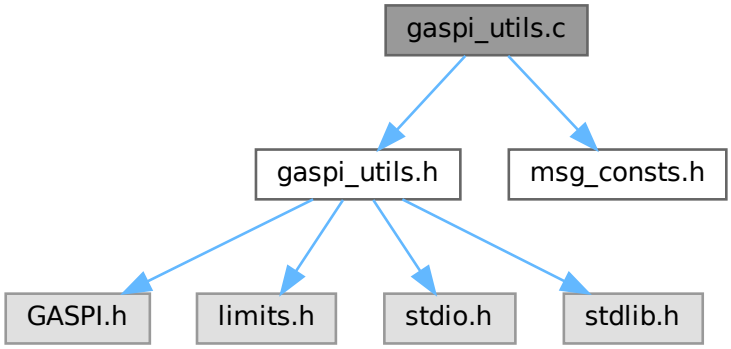
```
00117
00118     // Odbierz liczbę słów kluczowych
00119     memcpy(num_keywords, dict_seg_ptr, sizeof(int));
00120
00121     // Odbierz słowa kluczowe
00122     char word[MAX_WORD_LEN] = {0};
00123     for (int i = 0; i < *num_keywords; i++)
00124     {
00125         strncpy(word, (char *) dict_seg_ptr + sizeof(int) + i * MAX_WORD_LEN,
00126                 MAX_WORD_LEN - 1);
00127         word[MAX_WORD_LEN - 1] = '\0'; // Zabezpieczenie
00128
00129         keywords[i] = strdup(word);
00130         if (!keywords[i])
00131             exit(EXIT_FAILURE);
00132         insert_word(word, i);
00133     }
00134 }
```

5.20 gaspi_utils.c File Reference

Utility functions for GASPI segment management.

```
#include "gaspi_utils.h"
#include "msg_consts.h"
```

Include dependency graph for gaspi_utils.c:



Functions

- void cleanup_gaspi_segments (void)

  Cleans up GASPI segments.
- void init_gaspi_segments (void)

  Initializes GASPI segments for dictionary and work data.

5.20.1 Detailed Description

Utility functions for GASPI segment management.

Author

Wiktor Szewczyk

Definition in file gaspi_utils.c.

5.20.2 Function Documentation

5.20.2.1 cleanup_gaspi_segments()

```
void cleanup_gaspi_segments (
            void  )
```

Cleans up GASPI segments.

Definition at line 24 of file gaspi_utils.c.

References DICT_SEGMENT_ID, and WORK_SEGMENT_ID.

5.20.2.2 init_gaspi_segments()

```
void init_gaspi_segments (
            void  )
```

Initializes GASPI segments for dictionary and work data.

Definition at line 10 of file gaspi_utils.c.

References DICT_SEGMENT_ID, GASPI_CHECK, MAX_DOC_SIZE, MAX_KEYWORDS, MAX_WORD_LEN, and WORK_SEGMENT_ID.

5.21 gaspi_utils.c

Go to the documentation of this file.

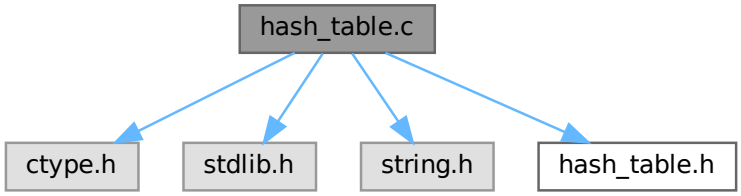
```
00001
00007 #include "gaspi_utils.h"
00008 #include "msg_consts.h"
00009
00010 void init_gaspi_segments(void)
00011 {
00012     gaspi_size_t dict_size = sizeof(int) + MAX_KEYWORDS * MAX_WORD_LEN;
00013     gaspi_size_t work_size = MAX_DOC_SIZE + sizeof(int);
00014
00015     // Segment dla słownika
00016     GASPI_CHECK(gaspi_segment_create(DICT_SEGMENT_ID, dict_size, GASPI_GROUP_ALL,
00017                                      GASPI_BLOCK, GASPI_ALLOC_DEFAULT));
00018
00019     // Segment dla pracy
00020     GASPI_CHECK(gaspi_segment_create(WORK_SEGMENT_ID, work_size, GASPI_GROUP_ALL,
00021                                      GASPI_BLOCK, GASPI_ALLOC_DEFAULT));
00022 }
00023
00024 void cleanup_gaspi_segments(void)
00025 {
00026     gaspi_segment_delete(DICT_SEGMENT_ID);
00027     gaspi_segment_delete(WORK_SEGMENT_ID);
00028 }
```

5.22 hash_table.c File Reference

Implementation of fixed-size chained hash table for keyword lookup.

```
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include "hash_table.h"
```

Include dependency graph for hash_table.c:



Functions

- void clear_hash_table (void)

  Frees all allocated hash table entries.
- int find_word (const char ∗word)

  Finds a word in the hash table.
- int hash_func (const char ∗s)

  Hash function for a word.
- void insert_word (const char ∗word, int index)

  Inserts a keyword into the hash table.
- void to_lower (char ∗str)

  Converts a string to lowercase in-place.

Variables

- Node ∗ hash_table [101]

  Global hash table structure.

5.22.1 Detailed Description

Implementation of fixed-size chained hash table for keyword lookup.

Author

Wiktor Szewczyk

Definition in file hash_table.c.

5.22.2 Function Documentation

5.22.2.1 clear_hash_table()

```
void clear_hash_table (
            void  )
```

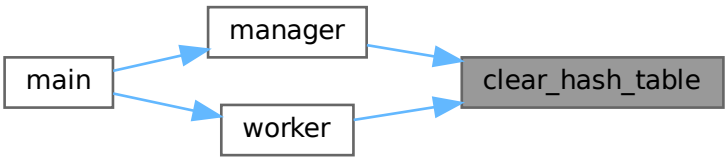
Frees all allocated hash table entries.

Definition at line 48 of file hash_table.c.

References HASH_SIZE, and hash_table.

Referenced by manager(), and worker().

Here is the caller graph for this function:



5.22.2.2 find_word()

```
int find_word (
            const char * word )
```

Finds a word in the hash table.

Parameters

word	Word to find

Returns

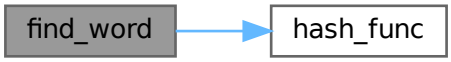
Index in dictionary, or -1 if not found

Definition at line 35 of file hash_table.c.

References hash_func(), and hash_table.

Referenced by classify_text().

Here is the call graph for this function:



Here is the caller graph for this function:

5.22.2.3 hash_func()

```
int hash_func (
            const char * s )
```

Hash function for a word.

Parameters

s	Word to hash

Returns

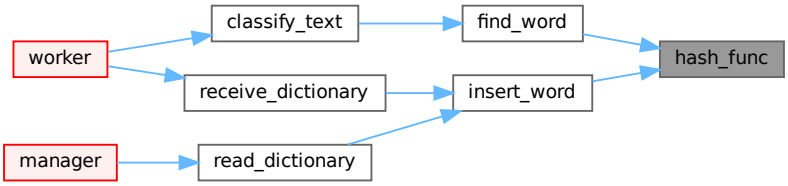
Index in hash table

Definition at line 15 of file hash_table.c.

References HASH_SIZE.

Referenced by find_word(), and insert_word().

Here is the caller graph for this function:



5.22.2.4 insert_word()

```
void insert_word (
            const char * word,
            int index )
```

Inserts a keyword into the hash table.

Parameters

word	Keyword string
index	Index in the dictionary

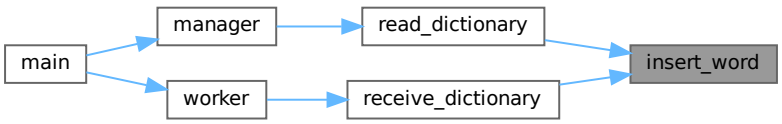
Definition at line 25 of file hash_table.c.

References hash_func(), and hash_table.

Referenced by read_dictionary(), and receive_dictionary().

Here is the call graph for this function:

Here is the caller graph for this function:



5.22.2.5 to_lower()

```
void to_lower (
            char * str )
```

Converts a string to lowercase in-place.

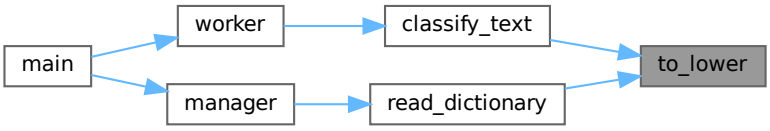
Parameters

str	String to lowercase

Definition at line 64 of file hash_table.c.

Referenced by classify_text(), and read_dictionary().

Here is the caller graph for this function:



5.22.3 Variable Documentation

5.22.3.1 hash_table

```
Node* hash_table[101]
```

Global hash table structure.

Definition at line 13 of file hash_table.c.

Referenced by clear_hash_table(), find_word(), and insert_word().

5.23 hash_table.c

Go to the documentation of this file.

```
00001
00007 #include <ctype.h>
00008 #include <stdlib.h>
00009 #include <string.h>
00010
00011 #include "hash_table.h"
00012
00013 Node *hash_table[HASH_SIZE];
00014
00015 int hash_func(const char *s)
00016 {
00017     int h = 0;
00018     for (int i = 0; s[i]; i++)
00019     {
00020         h = (h * 31 + s[i]) % HASH_SIZE;
00021     }
00022     return h;
00023 }
00024
00025 void insert_word(const char *word, int index)
00026 {
00027     int h = hash_func(word);
00028     Node *new_node = malloc(sizeof(Node));
00029     new_node->word = strdup(word);
00030     new_node->index = index;
00031     new_node->next = hash_table[h];
00032     hash_table[h] = new_node;
00033 }
00034
00035 int find_word(const char *word)
00036 {
00037     int h = hash_func(word);
00038     Node *cur = hash_table[h];
00039     while (cur)
00040     {
00041         if (strcmp(cur->word, word) == 0)
00042             return cur->index;
00043         cur = cur->next;
00044     }
00045     return -1;
00046 }
00047
00048 void clear_hash_table(void)
00049 {
00050     for (int i = 0; i < HASH_SIZE; i++)
00051     {
00052         Node *cur = hash_table[i];
00053         while (cur)
00054         {
00055             Node *tmp = cur;
00056             cur = cur->next;
00057             free(tmp->word);
00058             free(tmp);
00059         }
00060         hash_table[i] = NULL;
00061     }
00062 }
00063
00064 void to_lower(char *str)
00065 {
00066     for (; *str; ++str)
00067         *str = tolower(*str);
00068 }
```

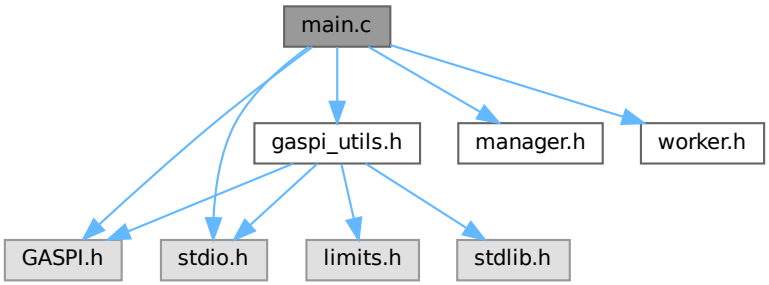
5.24 main.c File Reference

Entry point for the GASPI-based document classifier.

```
#include <GASPI.h>
#include <stdio.h>
#include "gaspi_utils.h"
#include "manager.h"
```

```
#include "worker.h"
```

Include dependency graph for main.c:



Functions

- int main (int argc, char ∗argv[ ])

5.24.1 Detailed Description

Entry point for the GASPI-based document classifier.

Author

Wiktor Szewczyk

Definition in file main.c.

5.24.2 Function Documentation

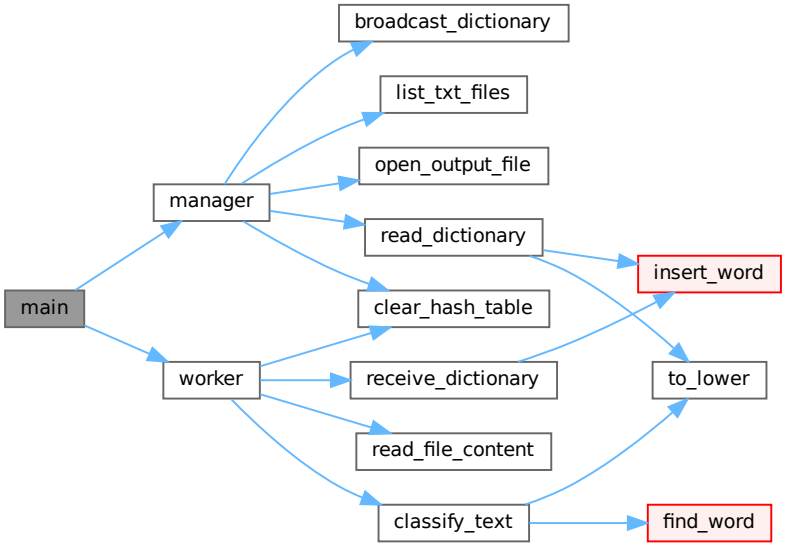
5.24.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Definition at line 14 of file main.c.

References GASPI_CHECK, manager(), and worker().

Here is the call graph for this function:



5.25 main.c

Go to the documentation of this file.

```
00001
00007 #include <GASPI.h>
00008 #include <stdio.h>
00009
00010 #include "gaspi_utils.h"
00011 #include "manager.h"
00012 #include "worker.h"
00013
00014 int main(int argc, char *argv[])
00015 {
00016     gaspi_rank_t rank = 0;
00017     gaspi_rank_t size = 0;
00018
00019     printf("[MAIN] Initializing GASPI...\n");
00020     GASPI_CHECK(gaspi_proc_init(GASPI_BLOCK));
00021     GASPI_CHECK(gaspi_proc_rank(&rank));
00022     GASPI_CHECK(gaspi_proc_num(&size));
00023
00024     printf("[MAIN] Process %d of %d started\n", rank, size);
00025
00026     if (argc != 4)
00027     {
00028         if (rank == 0)
00029         {
00030             fprintf(stderr, "Usage: %s <input_dir> <dict_file> <output_file>\n",
00031                     argv[0]);
00032             printf("[MAIN] Manager: Invalid arguments provided\n");
00033         }
00034         gaspi_proc_term(GASPI_BLOCK);
00035         return 1;
00036     }
00037
00038     if (rank == 0)
00039     {
00040         printf("[MAIN] Starting as MANAGER\n");
00041         printf("[MAIN] Input dir: %s\n", argv[1]);
00042         printf("[MAIN] Dictionary: %s\n", argv[2]);
00043         printf("[MAIN] Output file: %s\n", argv[3]);
00044         manager(argv[1], argv[2], argv[3], size);
00045     }
00046     else
```

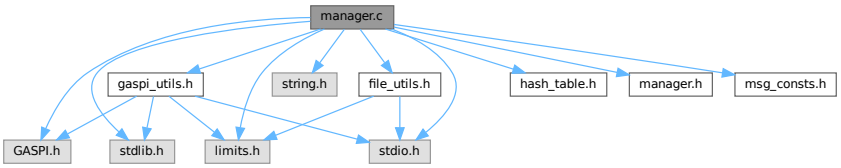
```
00047     {
00048         printf("[MAIN] Starting as WORKER %d\n", rank);
00049         worker();
00050     }
00051
00052     printf("[MAIN] Process %d terminating...\n", rank);
00053     gaspi_proc_term(GASPI_BLOCK);
00054     printf("[MAIN] Process %d terminated successfully\n", rank);
00055     return 0;
00056 }
```

5.26 manager.c File Reference

Orchestrates the classification by managing worker coordination and output collection.

```
#include <GASPI.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "file_utils.h"
#include "gaspi_utils.h"
#include "hash_table.h"
#include "manager.h"
#include "msg_consts.h"
```

Include dependency graph for manager.c:



Functions

- void manager (const char *input_dir, const char *dict_file, const char *output_file, int processes_num)

  Entry point for the manager process.

5.26.1 Detailed Description

Orchestrates the classification by managing worker coordination and output collection.

Author

Wiktor Szewczyk

Definition in file manager.c.

5.26.2 Function Documentation

5.26.2.1 manager()

```
void manager (
            const char * input_dir,
            const char * dict_file,
            const char * output_file,
            int processes_num )
```

Entry point for the manager process.

The manager is responsible for:

- Reading and broadcasting the dictionary
- Scanning the input directory for text files
- Distributing work to worker processes
- Collecting and writing classification results

Parameters

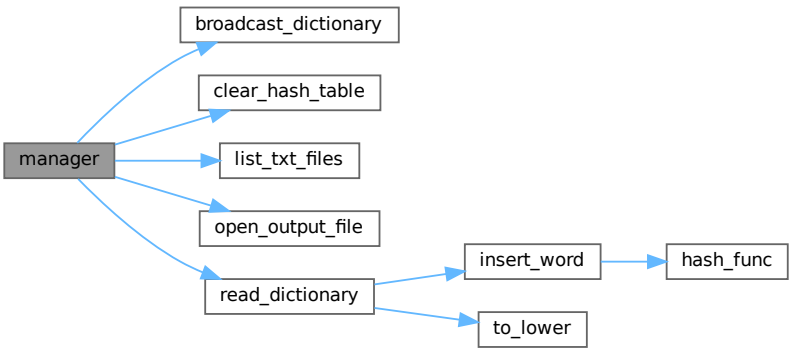
input_dir	Path to directory containing .txt files
dict_file	Path to the dictionary file
output_file	Path where results will be written
processes_num	Number of GASPI processes

Definition at line 20 of file manager.c.

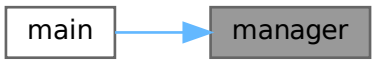
References broadcast_dictionary(), clear_hash_table(), DICT_SEGMENT_ID, GASPI_CHECK, list_txt_files(), MAX_FILES, MAX_KEYWORDS, MAX_WORD_LEN, open_output_file(), read_dictionary(), RESULT_OFFSET, RESULT_SEGMENT_ID, SHUTDOWN_NOTIF, WORK_AVAILABLE_NOTIF, WORK_DONE_NOTIF, WORK_NOTIF_ID, WORK_OFFSET, WORK_SEGMENT_ID, and WORKER_DATA_SIZE.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.27 manager.c

Go to the documentation of this file.

```
00001
00008 #include <GASPI.h>
00009 #include <limits.h>
00010 #include <stdio.h>
00011 #include <stdlib.h>
00012 #include <string.h>
00013
00014 #include "file_utils.h"
00015 #include "gaspi_utils.h"
00016 #include "hash_table.h"
00017 #include "manager.h"
00018 #include "msg_consts.h"
00019
00020 void manager(const char *input_dir, const char *dict_file, const char *output_file,
00021              int processes_num)
00022 {
00023     printf("[MANAGER] Starting with %d processes\n", processes_num);
00024
00025     gaspi_size_t dict_size = sizeof(int) + MAX_KEYWORDS * MAX_WORD_LEN;
00026     gaspi_size_t work_size = processes_num * PATH_MAX;
00027     gaspi_size_t result_size = processes_num * WORKER_DATA_SIZE;
00028
00029     printf("[MANAGER] Creating GASPI segments...\n");
00030     GASPI_CHECK(gaspi_segment_create(DICT_SEGMENT_ID, dict_size, GASPI_GROUP_ALL,
00031                                      GASPI_BLOCK, GASPI_ALLOC_DEFAULT));
00032
00033     GASPI_CHECK(gaspi_segment_create(WORK_SEGMENT_ID, work_size, GASPI_GROUP_ALL,
00034                                      GASPI_BLOCK, GASPI_ALLOC_DEFAULT));
00035
00036     GASPI_CHECK(gaspi_segment_create(RESULT_SEGMENT_ID, result_size, GASPI_GROUP_ALL,
00037                                      GASPI_BLOCK, GASPI_ALLOC_DEFAULT));
00038     printf("[MANAGER] GASPI segments created successfully\n");
00039
00040     char *keywords[MAX_KEYWORDS] = {0};
00041     int num_keywords = read_dictionary(dict_file, keywords, MAX_KEYWORDS);
00042     if (num_keywords < 0)
00043     {
00044         perror("Failed to read dictionary");
00045         gaspi_proc_term(GASPI_BLOCK);
00046         exit(1);
00047     }
00048     printf("[MANAGER] Dictionary loaded: %d keywords\n", num_keywords);
00049
00050     printf("[MANAGER] Broadcasting dictionary to workers...\n");
00051     broadcast_dictionary(keywords, num_keywords);
00052
00053     char files[MAX_FILES][PATH_MAX] = {0};
00054     int file_count = list_txt_files(input_dir, files, MAX_FILES);
00055     if (file_count < 0)
00056     {
00057         fprintf(stderr, "Error reading directory: %s\n", input_dir);
00058         gaspi_proc_term(GASPI_BLOCK);
00059         exit(1);
00060     }
00061     printf("[MANAGER] Found %d files to process\n", file_count);
00062
00063     FILE *out = open_output_file(output_file);
00064     if (!out)
00065     {
00066         perror("Failed to open output file");
00067         gaspi_proc_term(GASPI_BLOCK);
```

```
00068         exit(1);
00069     }
00070
00071     fprintf(out, "%-12s:", "dictionary");
00072     for (int i = 0; i < num_keywords; i++)
00073         fprintf(out, " %s", keywords[i]);
00074     fprintf(out, "\n");
00075
00076     int current_file = 0;
00077     int done_workers = 0;
00078
00079     gaspi_pointer_t work_seg_ptr;
00080     gaspi_pointer_t result_seg_ptr;
00081     GASPI_CHECK(gaspi_segment_ptr(WORK_SEGMENT_ID, &work_seg_ptr));
00082     GASPI_CHECK(gaspi_segment_ptr(RESULT_SEGMENT_ID, &result_seg_ptr));
00083
00084     printf("[MANAGER] Sending initial tasks to workers...\n");
00085     for (gaspi_rank_t rank = 1; rank < processes_num; rank++)
00086     {
00087         if (current_file < file_count)
00088         {
00089             gaspi_offset_t work_offset = WORK_OFFSET(rank);
00090             strcpy((char *) work_seg_ptr + work_offset, files[current_file]);
00091             printf("[MANAGER] Sending file '%s' to worker %d\n", files[current_file],
00092                    rank);
00093             current_file++;
00094             GASPI_CHECK(gaspi_write_notify(
00095                 WORK_SEGMENT_ID, work_offset, rank, WORK_SEGMENT_ID, work_offset,
00096                 PATH_MAX, WORK_NOTIF_ID, WORK_AVAILABLE_NOTIF, 1, GASPI_BLOCK));
00097         }
00098     }
00099
00100     printf("[MANAGER] Waiting for worker results...\n");
00101     while (done_workers < processes_num - 1)
00102     {
00103         gaspi_notification_id_t notif_id;
00104         gaspi_return_t ret = gaspi_notify_waitsome(
00105             RESULT_SEGMENT_ID, 1, processes_num - 1, &notif_id, GASPI_TEST);
00106         if (ret == GASPI_SUCCESS)
00107         {
00108             gaspi_notification_t notif_val;
00109             GASPI_CHECK(gaspi_notify_reset(RESULT_SEGMENT_ID, notif_id, &notif_val));
00110
00111             if (notif_val == WORK_DONE_NOTIF)
00112             {
00113                 gaspi_rank_t rank = notif_id;
00114                 gaspi_offset_t result_offset = RESULT_OFFSET(rank);
00115
00116                 int *vec = (int *) ((char *) result_seg_ptr + result_offset);
00117                 char *fname = (char *) ((char *) result_seg_ptr + result_offset +
00118                                         MAX_KEYWORDS * sizeof(int));
00119
00120                 printf("[MANAGER] Received results from worker %d for file '%s'\n",
00121                        rank, strrchr(fname, '/') ? strrchr(fname, '/') + 1 : fname);
00122
00123                 fprintf(out, "%-12s:",
00124                         strrchr(fname, '/') ? strrchr(fname, '/') + 1 : fname);
00125                 for (int i = 0; i < num_keywords; i++)
00126                     fprintf(out, " %d", vec[i]);
00127                 fprintf(out, "\n");
00128
00129                 if (current_file < file_count)
00130                 {
00131                     gaspi_offset_t work_offset = WORK_OFFSET(rank);
00132                     strcpy((char *) work_seg_ptr + work_offset, files[current_file]);
00133                     printf("[MANAGER] Sending next file '%s' to worker %d\n",
00134                            files[current_file], rank);
00135                     current_file++;
00136                     GASPI_CHECK(gaspi_write_notify(
00137                         WORK_SEGMENT_ID, work_offset, rank, WORK_SEGMENT_ID,
00138                         work_offset, PATH_MAX, WORK_NOTIF_ID, WORK_AVAILABLE_NOTIF, 1,
00139                         GASPI_BLOCK));
00140                 }
00141                 else
00142                 {
00143                     gaspi_offset_t work_offset = WORK_OFFSET(rank);
00144                     strcpy((char *) work_seg_ptr + work_offset, "");
00145                     printf("[MANAGER] Sending shutdown signal to worker %d\n", rank);
00146                     GASPI_CHECK(gaspi_write_notify(
00147                         WORK_SEGMENT_ID, work_offset, rank, WORK_SEGMENT_ID,
00148                         work_offset, 1, WORK_NOTIF_ID, SHUTDOWN_NOTIF, 1, GASPI_BLOCK));
00149                     done_workers++;
00150                 }
00151             }
00152         }
00153     }
00154
```

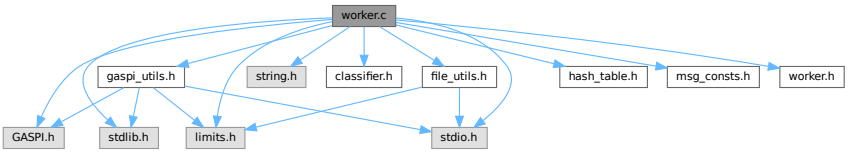
```
00155     printf("[MANAGER] All workers finished, cleaning up...\n");
00156     fclose(out);
00157     for (int i = 0; i < num_keywords; i++)
00158         free(keywords[i]);
00159     clear_hash_table();
00160
00161     gaspi_segment_delete(DICT_SEGMENT_ID);
00162     gaspi_segment_delete(WORK_SEGMENT_ID);
00163     gaspi_segment_delete(RESULT_SEGMENT_ID);
00164     printf("[MANAGER] Cleanup complete\n");
00165 }
```

5.28 worker.c File Reference

Logic for a worker process participating in distributed document classification.

```
#include <GASPI.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "classifier.h"
#include "file_utils.h"
#include "gaspi_utils.h"
#include "hash_table.h"
#include "msg_consts.h"
#include "worker.h"
```

Include dependency graph for worker.c:



Functions

- void worker (void)

  Entry point for each GASPI worker process.

5.28.1 Detailed Description

Logic for a worker process participating in distributed document classification.

Author

Wiktor Szewczyk

Definition in file worker.c.

5.28.2 Function Documentation

5.28.2.1 worker()

```
void worker (
            void  )
```

Entry point for each GASPI worker process.

Each worker:

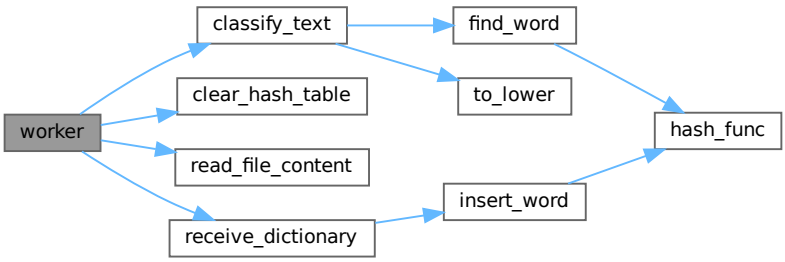
- Receives the dictionary from the manager
- Processes assigned documents
- Sends back feature vectors to the manager

Definition at line 21 of file worker.c.

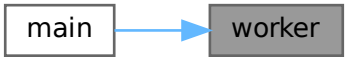
References classify_text(), clear_hash_table(), GASPI_CHECK, MAX_DOC_SIZE, MAX_KEYWORDS, read_file_content(), receive_dictionary(), RESULT_OFFSET, RESULT_SEGMENT_ID, SHUTDOWN_NOTIF, WORK_AVAILABLE_NOTIF, WORK_DONE_NOTIF, WORK_NOTIF_ID, WORK_OFFSET, and WORK_SEGMENT_ID.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.29 worker.c

Go to the documentation of this file.

```
00001
00008 #include <GASPI.h>
00009 #include <limits.h>
00010 #include <stdio.h>
00011 #include <stdlib.h>
00012 #include <string.h>
00013
00014 #include "classifier.h"
00015 #include "file_utils.h"
00016 #include "gaspi_utils.h"
00017 #include "hash_table.h"
00018 #include "msg_consts.h"
00019 #include "worker.h"
00020
00021 void worker(void)
00022 {
00023     gaspi_rank_t my_rank;
00024     GASPI_CHECK(gaspi_proc_rank(&my_rank));
00025
00026     printf("[WORKER %d] Starting up...\n", my_rank);
00027
00028     char *keywords[MAX_KEYWORDS] = {0};
00029     int num_keywords;
00030     printf("[WORKER %d] Waiting for dictionary...\n", my_rank);
00031     receive_dictionary(keywords, &num_keywords);
00032     printf("[WORKER %d] Received dictionary with %d keywords\n", my_rank, num_keywords);
00033
00034     int vec[MAX_KEYWORDS] = {0};
00035     char fname[PATH_MAX] = {0};
00036     char buffer[MAX_DOC_SIZE] = {0};
00037
00038     gaspi_pointer_t work_seg_ptr;
00039     gaspi_pointer_t result_seg_ptr;
00040     GASPI_CHECK(gaspi_segment_ptr(WORK_SEGMENT_ID, &work_seg_ptr));
00041     GASPI_CHECK(gaspi_segment_ptr(RESULT_SEGMENT_ID, &result_seg_ptr));
00042
00043     gaspi_offset_t my_work_offset = WORK_OFFSET(my_rank);
00044     gaspi_offset_t my_result_offset = RESULT_OFFSET(my_rank);
00045
00046     printf("[WORKER %d] Ready for work (work_offset=%lu, result_offset=%lu)\n", my_rank,
00047            my_work_offset, my_result_offset);
00048
00049     int files_processed = 0;
00050     while (1)
00051     {
00052         printf("[WORKER %d] Waiting for task...\n", my_rank);
00053         gaspi_notification_id_t notif_id;
00054         GASPI_CHECK(gaspi_notify_waitsome(WORK_SEGMENT_ID, WORK_NOTIF_ID, 1, &notif_id,
00055                                           GASPI_BLOCK));
00056
00057         gaspi_notification_t notif_val;
00058         GASPI_CHECK(gaspi_notify_reset(WORK_SEGMENT_ID, notif_id, &notif_val));
00059
00060         if (notif_val == SHUTDOWN_NOTIF)
00061         {
00062             printf("[WORKER %d] Received shutdown signal, processed %d files total\n",
00063                    my_rank, files_processed);
00064             break;
00065         }
00066
00067         if (notif_val == WORK_AVAILABLE_NOTIF)
00068         {
00069             strcpy(fname, (char *) work_seg_ptr + my_work_offset);
00070             printf("[WORKER %d] Processing file: %s\n", my_rank, fname);
00071
00072             if (read_file_content(fname, buffer, sizeof(buffer)) < 0)
00073             {
00074                 printf("[WORKER %d] ERROR: Failed to read file %s\n", my_rank, fname);
00075                 memset(vec, 0, sizeof(vec));
00076             }
00077             else
00078             {
00079                 classify_text(buffer, vec, num_keywords);
00080                 printf("[WORKER %d] Classification complete for %s\n", my_rank, fname);
00081             }
00082
00083             int *vec_ptr = (int *) ((char *) result_seg_ptr + my_result_offset);
00084             char *fname_ptr = (char *) ((char *) result_seg_ptr + my_result_offset +
00085                                         MAX_KEYWORDS * sizeof(int));
00086
00087             memcpy(vec_ptr, vec, MAX_KEYWORDS * sizeof(int));
00088             strcpy(fname_ptr, fname);
```

```
00089
00090             printf("[WORKER %d] Sending results back to manager\n", my_rank);
00091             GASPI_CHECK(gaspi_write_notify(RESULT_SEGMENT_ID, my_result_offset, 0,
00092                                            RESULT_SEGMENT_ID, my_result_offset,
00093                                            MAX_KEYWORDS * sizeof(int) + PATH_MAX,
00094                                            my_rank, WORK_DONE_NOTIF, 1, GASPI_BLOCK));
00095             files_processed++;
00096         }
00097     }
00098
00099     printf("[WORKER %d] Cleaning up...\n", my_rank);
00100     for (int i = 0; i < num_keywords; i++)
00101         free(keywords[i]);
00102     clear_hash_table();
00103     printf("[WORKER %d] Shutdown complete\n", my_rank);
00104 }
```

Index