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 MPI, 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 MPICH to classify text documents into feature vectors. It uses hashing to match words against a shared dictionary and distributes processing using MPI.

It currently supports only `.txt` documents.

1.2 Architecture

- The **manager process**:
  - reads the dictionary file (one keyword per line),
  - broadcasts it to all worker processes,
  - scans the input directory for `.txt` files,
  - distributes file paths to workers,
  - receives classified vectors and writes them to the output file.
- Each **worker process**:
  - receives the dictionary,
  - receives a file path from the manager,
  - 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,
  - sends the result back to the manager.

1.3 Flow Diagram

1.4 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/install_mpich.sh       # Script for downloading, building and installing MPICH locally
|- scripts/gen_data.py            # Script for generating random data (see ./scripts/gen_data.py --help for
     more)
|- Makefile                       # Entry point Makefile
|- CMakeLists.txt                 # Entry point CMake
```

1.5 Requirements

- GCC 12+
- GNU Make
- MPICH 4.3.0+
- (Optional) `Doxygen 1.9.8+` + `Doxygen-Awesome-CSS`
- (Optional) `Criterion 2.4.1+` for unit testing

1.6 Installing MPICH Locally

```
./scripts/install_mpich.sh
```

This script installs MPICH 4.3.0 into a local `.mpich/` directory without system-wide changes.

1.7 Setting up MPICH path

Before building or running the project, you must set the `MPICH_TARGET_DIR` environment variable to point to the root directory of your MPICH installation:

```
export MPICH_TARGET_DIR=/path/to/mpich
```

1.8 Build

```
make build
```

Builds the executable at:

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

1.9 Running the Classifier

```
make run MPI_FLAGS='-f nodes -n 8' RUN_FLAGS='input/ dict.txt out.txt'
```

- `MPI_FLAGS`: passed to `mpiexec` (default: `-f nodes -n 16`)
- `RUN_FLAGS`: arguments for `ddc` (default: `./example/input/ ./example/dict.txt ./example/output/re`

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
- `MPICH`
- `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:

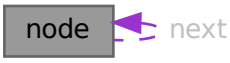
Chapter 4

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 25 of file hash_table.h.

4.1.2.2 next

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

Next node in chain.

Definition at line 26 of file hash_table.h.

4.1.2.3 word

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

Keyword string.

Definition at line 24 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

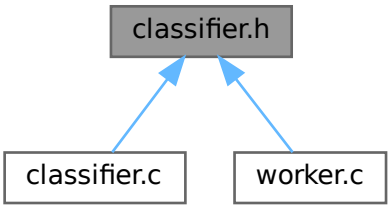
File Documentation

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

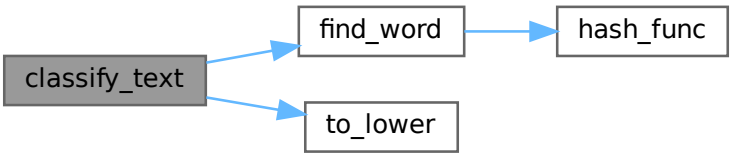
text	Input document text
result	Output vector (must be zeroed and sized to keyword_count)
keyword_count	Number of keywords in dictionary

Definition at line 13 of file classifier.c.

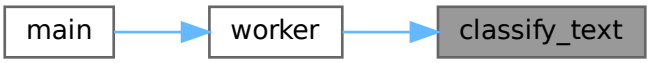
References CLASSIFY_TOKENS, find_word(), MAX_DOC_SIZE, and to_lower().

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.

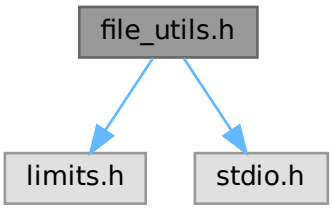
```
00001
00007 #ifndef CLASSIFIER_H
00008 #define CLASSIFIER_H
00009
00011 #define CLASSIFY_TOKENS " \n\t.,;:!?()[]{}\"'-"
00012
00023 void classify_text(const char *text, int result[], int keyword_count);
00024
00025 #endif
00026
```

5.4 file_utils.h File Reference

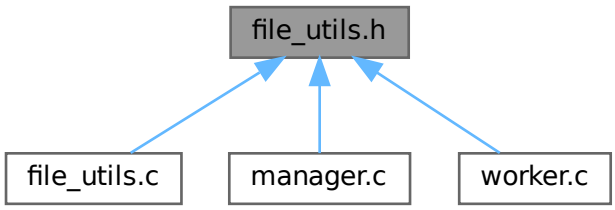
Utility functions for files operations.

```
#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 83 of file file_utils.c.

References 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

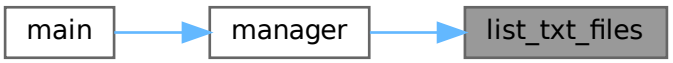
Returns

Number of files found, or -1 on error

Definition at line 18 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

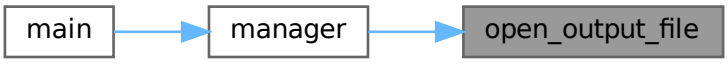
Returns

FILE* handle or NULL on failure

Definition at line 52 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

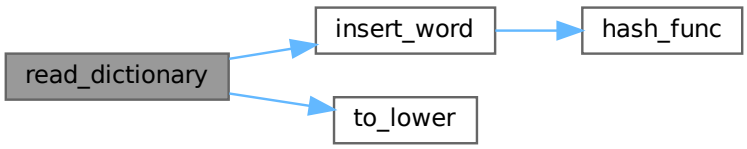
Number of keywords read, or -1 on error

Definition at line 57 of file file_utils.c.

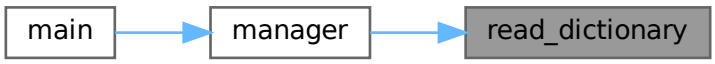
References insert_word(), MAX_WORD_LEN, and to_lower().

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 40 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

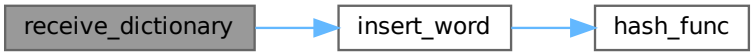
keywords	Output array of keyword strings
num_keywords	Output number of keywords

Definition at line 92 of file file_utils.c.

References 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

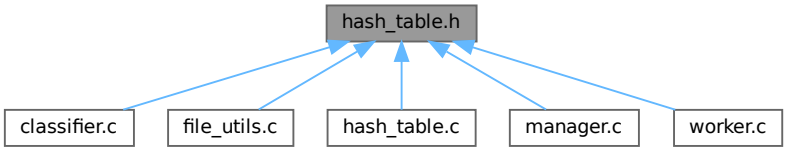
Go to the documentation of this file.

```
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
00068
```

5.6 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.

5.6.1 Detailed Description

Simple fixed-size chained hash table for word lookup.

Author

Wiktor Szewczyk

Definition in file hash_table.h.

5.6.2 Macro Definition Documentation

5.6.2.1 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.

5.6.3 Typedef Documentation

5.6.3.1 Node

```
typedef struct node Node
```

5.6.4 Function Documentation

5.6.4.1 clear_hash_table()

```
void clear_hash_table (
            void  )
```

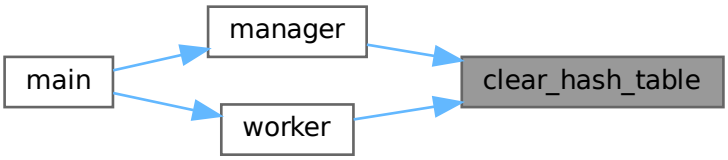
Frees all allocated hash table entries.

Definition at line 49 of file hash_table.c.

References HASH_SIZE, and hash_table.

Referenced by manager(), and worker().

Here is the caller graph for this function:



5.6.4.2 find_word()

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

Finds a word in the hash table.

Parameters

word	Word to find

Returns

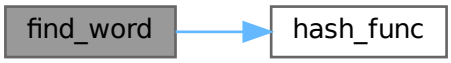
Index in dictionary, or -1 if not found

Definition at line 36 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:

5.6.4.3 hash_func()

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

Hash function for a word.

Parameters

s	Word to hash

Returns

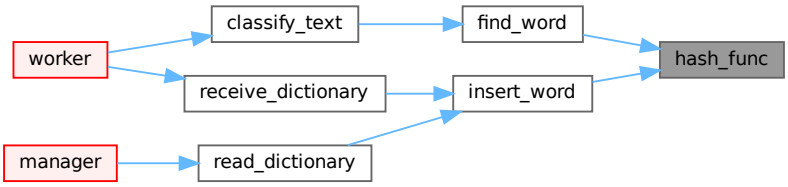
Index in hash table

Definition at line 16 of file hash_table.c.

References HASH_SIZE.

Referenced by find_word(), and insert_word().

Here is the caller graph for this function:



5.6.4.4 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

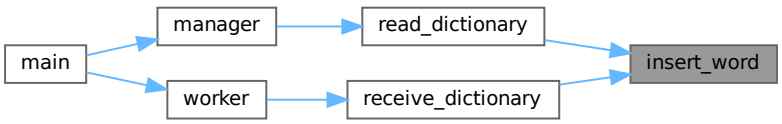
Definition at line 26 of file hash_table.c.

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:



5.6.4.5 to_lower()

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

Converts a string to lowercase in-place.

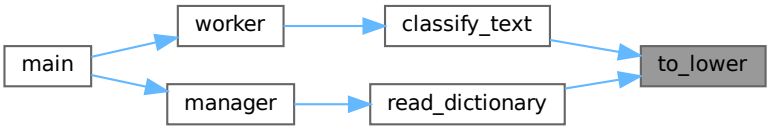
Parameters

str	String to lowercase

Definition at line 65 of file hash_table.c.

Referenced by classify_text(), and read_dictionary().

Here is the caller graph for this function:



5.6.5 Variable Documentation

5.6.5.1 hash_table

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

Global hash table structure.

Definition at line 14 of file hash_table.c.

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

5.7 hash_table.h

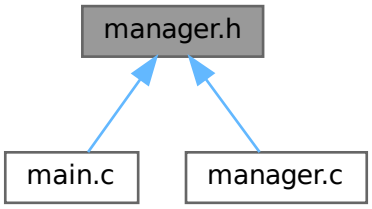
Go to the documentation of this file.

```
00001
00007 #ifndef HASH_TABLE_H
00008 #define HASH_TABLE_H
00009
00017 #define HASH_SIZE 101
00018
00023 typedef struct node {
00024     char *word;
00025     int index;
00026     struct node *next;
00027 } Node;
00028
00030 extern Node *hash_table[HASH_SIZE];
00031
00038 int hash_func(const char *s);
00039
00046 void insert_word(const char *word, int index);
00047
00054 int find_word(const char *word);
00055
00059 void clear_hash_table(void);
00060
00066 void to_lower(char *str);
00067
00068 #endif
00069
```

5.8 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 size)

  Entry point for the manager process.

5.8.1 Detailed Description

Interface for the manager process.

Author

Wiktor Szewczyk

Definition in file manager.h.

5.8.2 Function Documentation

5.8.2.1 manager()

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

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

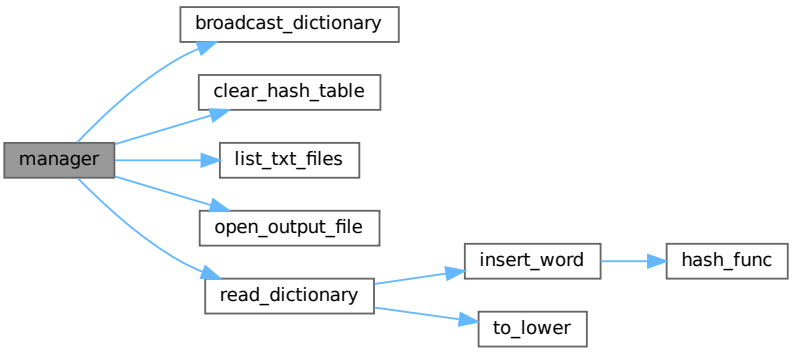
input_dir	Path to directory containing .txt files
dict_file	Path to the dictionary file
output_file	Path where results will be written
size	Number of MPI processes

Definition at line 17 of file manager.c.

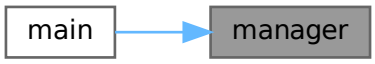
References broadcast_dictionary(), clear_hash_table(), DONE_MSG, FILE_MSG, list_txt_files(), MAX_FILES, MAX_KEYWORDS, open_output_file(), read_dictionary(), REQUEST_MSG, and VEC_MSG.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9 manager.h

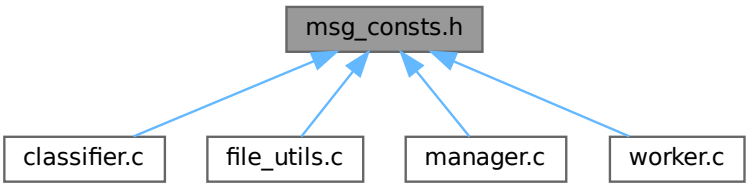
Go to the documentation of this file.

```
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, int size);
00025
00026 #endif
00027
```

5.10 msg_consts.h File Reference

Constants for MPI messaging and buffer sizes.

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



Macros

- #define DONE_MSG 3

  Worker signals completion.
- #define FILE_MSG 1

  Filename message.
- #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 REQUEST_MSG 0

  MPI message tags.
- #define VEC_MSG 2

  Feature vector message.

5.10.1 Detailed Description

Constants for MPI messaging and buffer sizes.

Author

Wiktor Szewczyk

Definition in file msg_consts.h.

5.10.2 Macro Definition Documentation

5.10.2.1 DONE_MSG

```
#define DONE_MSG 3
```

Worker signals completion.

Definition at line 26 of file msg_consts.h.

5.10.2.2 FILE_MSG

```
#define FILE_MSG 1
```

Filename message.

Definition at line 24 of file msg_consts.h.

5.10.2.3 MAX_DOC_SIZE

```
#define MAX_DOC_SIZE 4096
```

Maximum size of a document in bytes.

Definition at line 11 of file msg_consts.h.

5.10.2.4 MAX_FILES

```
#define MAX_FILES 1024
```

Maximum number of files.

Definition at line 20 of file msg_consts.h.

5.10.2.5 MAX_KEYWORDS

```
#define MAX_KEYWORDS 256
```

Maximum number of dictionary keywords.

Definition at line 14 of file msg_consts.h.

5.10.2.6 MAX_WORD_LEN

```
#define MAX_WORD_LEN 64
```

Maximum length of a single keyword.

Definition at line 17 of file msg_consts.h.

5.10.2.7 REQUEST_MSG

```
#define REQUEST_MSG 0
```

MPI message tags.

Worker requests a new task

Definition at line 23 of file msg_consts.h.

5.10.2.8 VEC_MSG

```
#define VEC_MSG 2
```

Feature vector message.

Definition at line 25 of file msg_consts.h.

5.11 msg_consts.h

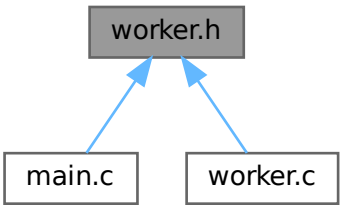
Go to the documentation of this file.

```
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 REQUEST_MSG 0
00024 #define FILE_MSG    1
00025 #define VEC_MSG     2
00026 #define DONE_MSG    3
00027
00028 #endif
00029
```

5.12 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 MPI worker process.

5.12.1 Detailed Description

Interface for worker process.

Author

Wiktor Szewczyk

Definition in file worker.h.

5.12.2 Function Documentation

5.12.2.1 worker()

```
void worker (
            void  )
```

Entry point for each MPI worker process.

Each worker:

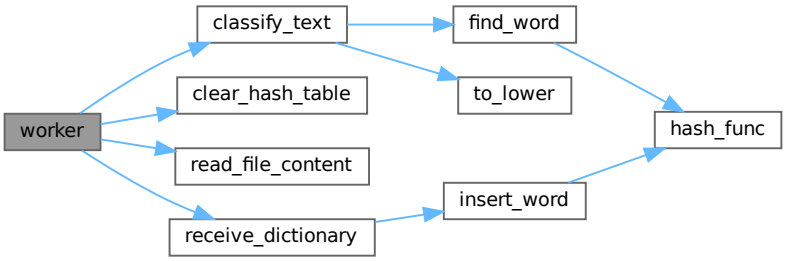
- Receives the dictionary from the manager
- Processes assigned documents
- Sends back feature vectors to the manager

Definition at line 19 of file worker.c.

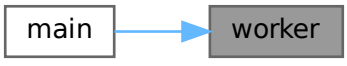
References classify_text(), clear_hash_table(), DONE_MSG, FILE_MSG, MAX_DOC_SIZE, MAX_KEYWORDS, read_file_content(), receive_dictionary(), REQUEST_MSG, and VEC_MSG.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.13 worker.h

Go to the documentation of this file.

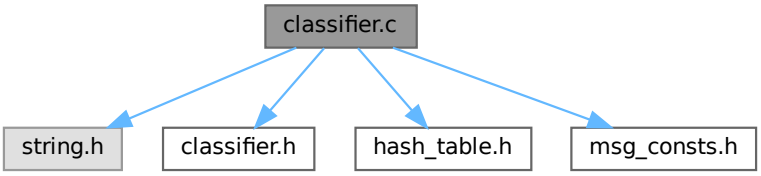
```
00001
00007 #ifndef WORKER_H
00008 #define WORKER_H
00009
00018 void worker(void);
00019
00020 #endif
00021
```

5.14 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.

5.14.1 Detailed Description

Implements document classification using tokenization and a hash table.

Author

Wiktor Szewczyk

Definition in file classifier.c.

5.14.2 Function Documentation

5.14.2.1 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

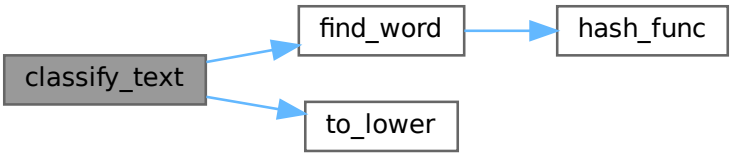
text	Input document text
result	Output vector (must be zeroed and sized to keyword_count)
keyword_count	Number of keywords in dictionary

Definition at line 13 of file classifier.c.

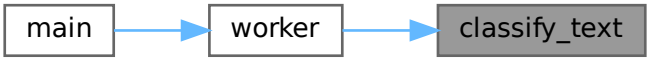
References CLASSIFY_TOKENS, find_word(), MAX_DOC_SIZE, and to_lower().

Referenced by worker().

Here is the call graph for this function:



Here is the caller graph for this function:



5.15 classifier.c

Go to the documentation of this file.

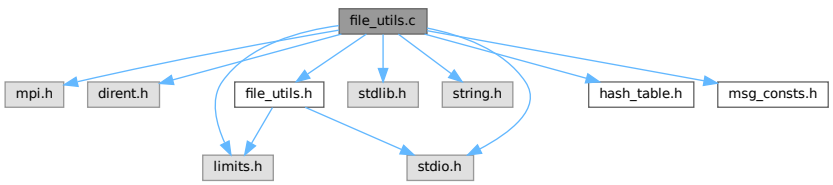
```
00001
00007 #include <string.h>
00008
00009 #include "classifier.h"
00010 #include "hash_table.h"
00011 #include "msg_consts.h"
00012
00013 void classify_text(const char *text, int result[], int keyword_count)
00014 {
00015     memset(result, 0, sizeof(int) * keyword_count);
00016
00017     char 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 }
00031
```

5.16 file_utils.c File Reference

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

```
#include <mpi.h>
#include <dirent.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "file_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.

5.16.1 Detailed Description

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

Author

Wiktor Szewczyk

Definition in file file_utils.c.

5.16.2 Function Documentation

5.16.2.1 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 83 of file file_utils.c.

References MAX_WORD_LEN.

Referenced by manager().

Here is the caller graph for this function:

5.16.2.2 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

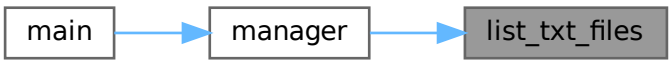
Returns

Number of files found, or -1 on error

Definition at line 18 of file file_utils.c.

Referenced by manager().

Here is the caller graph for this function:



5.16.2.3 open_output_file()

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

Opens the result output file for writing.

Parameters

output_path	Output file path

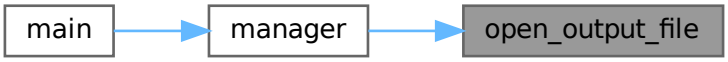
Returns

FILE∗ handle or NULL on failure

Definition at line 52 of file file_utils.c.

Referenced by manager().

Here is the caller graph for this function:



5.16.2.4 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

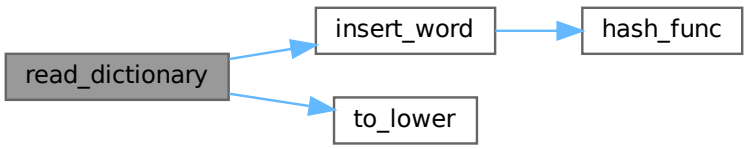
Number of keywords read, or -1 on error

Definition at line 57 of file file_utils.c.

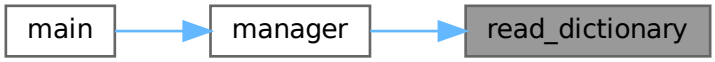
References insert_word(), MAX_WORD_LEN, and to_lower().

Referenced by manager().

Here is the call graph for this function:



Here is the caller graph for this function:



5.16.2.5 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 40 of file file_utils.c.

Referenced by worker().

Here is the caller graph for this function:

5.16.2.6 receive_dictionary()

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

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

Parameters

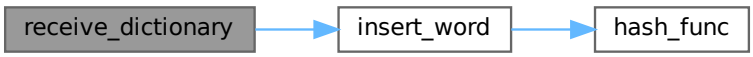
keywords	Output array of keyword strings
num_keywords	Output number of keywords

Definition at line 92 of file file_utils.c.

References 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:

5.17 file_utils.c

Go to the documentation of this file.

```
00001
00007 #include <mpi.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 "hash_table.h"
00016 #include "msg_consts.h"
00017
00018 int list_txt_files(const char *dir_path, char files[][PATH_MAX], int max_files)
00019 {
00020     DIR *dir = opendir(dir_path);
00021     if (!dir)
00022         return -1;
00023
00024     struct dirent *entry;
00025     int count = 0;
00026
00027     while ((entry = readdir(dir)) != NULL && count < max_files)
00028     {
00029         if (strstr(entry->d_name, ".txt"))
```

```
00030         {
00031             snprintf(files[count], PATH_MAX, "%s/%s", dir_path, entry->d_name);
00032             count++;
00033         }
00034     }
00035
00036     closedir(dir);
00037     return count;
00038 }
00039
00040 int read_file_content(const char *filename, char *buffer, size_t bufsize)
00041 {
00042     FILE *f = fopen(filename, "r");
00043     if (!f)
00044         return -1;
00045
00046     size_t n = fread(buffer, 1, bufsize - 1, f);
00047     buffer[n] = '\0';
00048     fclose(f);
00049     return 0;
00050 }
00051
00052 FILE *open_output_file(const char *output_path)
00053 {
00054     return fopen(output_path, "w");
00055 }
00056
00057 int read_dictionary(const char *dict_path, char *keywords[], int max_keywords)
00058 {
00059     FILE *df = fopen(dict_path, "r");
00060     if (!df)
00061         return -1;
00062
00063     int count = 0;
00064     char word[MAX_WORD_LEN] = {0};
00065
00066     while (fscanf(df, "%63s", word) == 1 && count < max_keywords)
00067     {
00068         to_lower(word);
00069         keywords[count] = strdup(word);
00070         if (!keywords[count])
00071         {
00072             fclose(df);
00073             return -2;
00074         }
00075         insert_word(word, count);
00076         count++;
00077     }
00078
00079     fclose(df);
00080     return count;
00081 }
00082
00083 void broadcast_dictionary(char *keywords[], int num_keywords)
00084 {
00085     MPI_Bcast(&num_keywords, 1, MPI_INT, 0, MPI_COMM_WORLD);
00086     for (int i = 0; i < num_keywords; i++)
00087     {
00088         MPI_Bcast(keywords[i], MAX_WORD_LEN, MPI_CHAR, 0, MPI_COMM_WORLD);
00089     }
00090 }
00091
00092 void receive_dictionary(char *keywords[], int *num_keywords)
00093 {
00094     MPI_Bcast(num_keywords, 1, MPI_INT, 0, MPI_COMM_WORLD);
00095     char word[MAX_WORD_LEN] = {0};
00096     for (int i = 0; i < *num_keywords; i++)
00097     {
00098         MPI_Bcast(word, MAX_WORD_LEN, MPI_CHAR, 0, MPI_COMM_WORLD);
00099         keywords[i] = strdup(word);
00100         if (!keywords[i])
00101             exit(EXIT_FAILURE);
00102         insert_word(word, i);
00103     }
00104 }
```

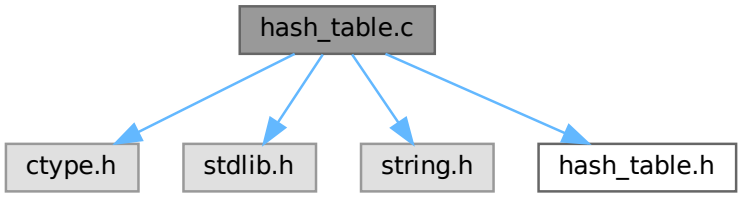
5.18 hash_table.c File Reference

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.

5.18.1 Detailed Description

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

Author

Wiktor Szewczyk

Definition in file hash_table.c.

5.18.2 Function Documentation

5.18.2.1 clear_hash_table()

```
void clear_hash_table (
            void  )
```

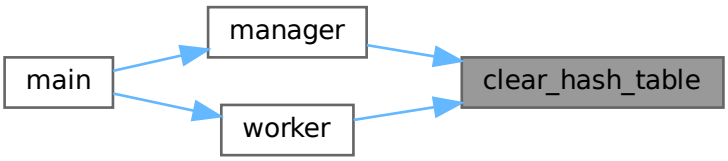
Frees all allocated hash table entries.

Definition at line 49 of file hash_table.c.

References HASH_SIZE, and hash_table.

Referenced by manager(), and worker().

Here is the caller graph for this function:



5.18.2.2 find_word()

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

Finds a word in the hash table.

Parameters

word	Word to find

Returns

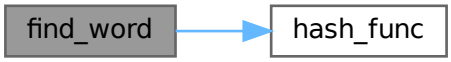
Index in dictionary, or -1 if not found

Definition at line 36 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:

5.18.2.3 hash_func()

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

Hash function for a word.

Parameters

s	Word to hash

Returns

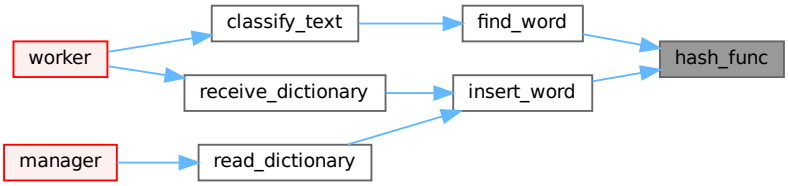
Index in hash table

Definition at line 16 of file hash_table.c.

References HASH_SIZE.

Referenced by find_word(), and insert_word().

Here is the caller graph for this function:



5.18.2.4 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

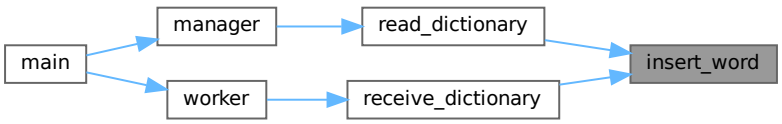
Definition at line 26 of file hash_table.c.

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:



5.18.2.5 to_lower()

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

Converts a string to lowercase in-place.

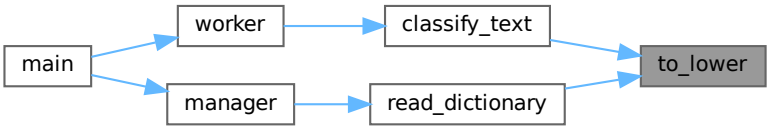
Parameters

str	String to lowercase

Definition at line 65 of file hash_table.c.

Referenced by classify_text(), and read_dictionary().

Here is the caller graph for this function:



5.18.3 Variable Documentation

5.18.3.1 hash_table

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

Global hash table structure.

Definition at line 14 of file hash_table.c.

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

5.19 hash_table.c

Go to the documentation of this file.

```
00001
00008 #include <ctype.h>
00009 #include <stdlib.h>
00010 #include <string.h>
00011
00012 #include "hash_table.h"
00013
00014 Node *hash_table[HASH_SIZE];
00015
00016 int hash_func(const char *s)
00017 {
00018     int h = 0;
00019     for (int i = 0; s[i]; i++)
00020     {
00021         h = (h * 31 + s[i]) % HASH_SIZE;
00022     }
00023     return h;
00024 }
00025
00026 void insert_word(const char *word, int index)
00027 {
00028     int h = hash_func(word);
00029     Node *new_node = malloc(sizeof(Node));
00030     new_node->word = strdup(word);
00031     new_node->index = index;
00032     new_node->next = hash_table[h];
00033     hash_table[h] = new_node;
00034 }
00035
00036 int find_word(const char *word)
00037 {
00038     int h = hash_func(word);
00039     Node *cur = hash_table[h];
00040     while (cur)
00041     {
00042         if (strcmp(cur->word, word) == 0)
00043             return cur->index;
00044         cur = cur->next;
00045     }
00046     return -1;
00047 }
00048
00049 void clear_hash_table(void)
00050 {
00051     for (int i = 0; i < HASH_SIZE; i++)
00052     {
00053         Node *cur = hash_table[i];
00054         while (cur)
00055         {
00056             Node *tmp = cur;
00057             cur = cur->next;
00058             free(tmp->word);
00059             free(tmp);
00060         }
00061         hash_table[i] = NULL;
00062     }
00063 }
00064
00065 void to_lower(char *str)
00066 {
00067     for (; *str; ++str)
00068         *str = tolower(*str);
00069 }
```

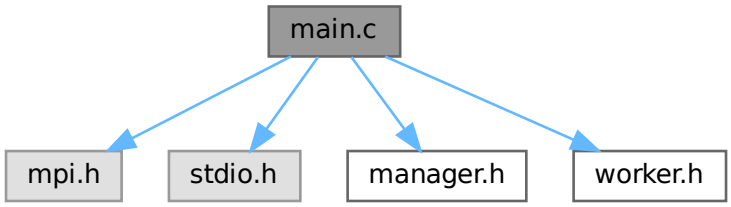
5.20 main.c File Reference

Entry point for the MPI-based document classifier.

```
#include <mpi.h>
#include <stdio.h>
#include "manager.h"
```

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

Include dependency graph for main.c:



Functions

- int main (int argc, char ∗argv[ ])

5.20.1 Detailed Description

Entry point for the MPI-based document classifier.

Author

Wiktor Szewczyk

Definition in file main.c.

5.20.2 Function Documentation

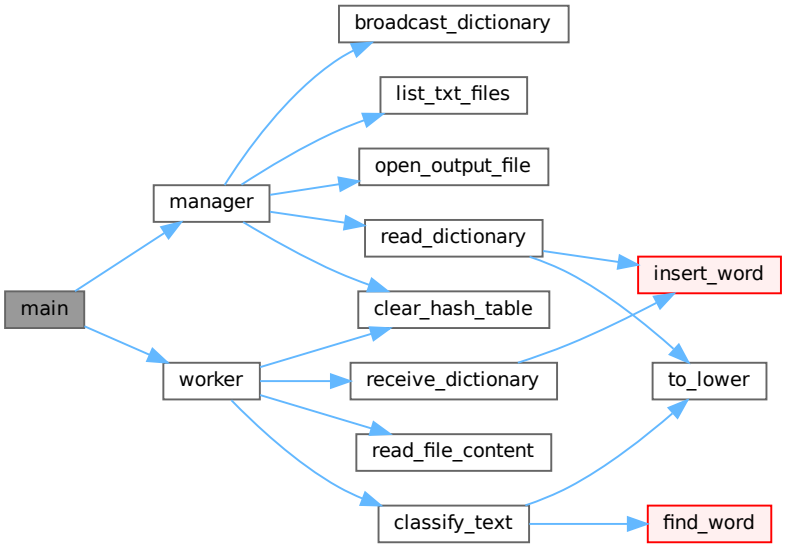
5.20.2.1 main()

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

Definition at line 13 of file main.c.

References manager(), and worker().

Here is the call graph for this function:



5.21 main.c

Go to the documentation of this file.

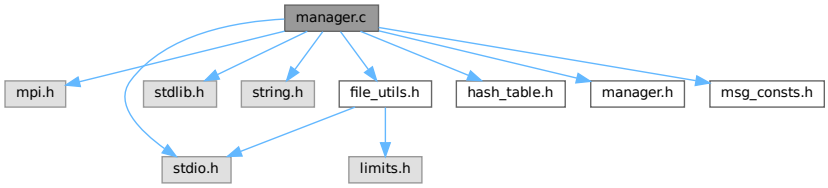
```
00001
00007 #include <mpi.h>
00008 #include <stdio.h>
00009
00010 #include "manager.h"
00011 #include "worker.h"
00012
00013 int main(int argc, char *argv[])
00014 {
00015     int rank = -1;
00016     int size = -1;
00017     MPI_Init(&argc, &argv);
00018     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00019     MPI_Comm_size(MPI_COMM_WORLD, &size);
00020
00021     if (argc != 4)
00022     {
00023         if (rank == 0)
00024             fprintf(stderr, "Usage: %s <input_dir> <dict_file> <output_file>\n",
00025                     argv[0]);
00026         MPI_Finalize();
00027         return 1;
00028     }
00029
00030     if (rank == 0)
00031         manager(argv[1], argv[2], argv[3], size);
00032     else
00033         worker();
00034
00035     MPI_Finalize();
00036     return 0;
00037 }
```

5.22 manager.c File Reference

Orchestrates the classification by managing worker coordination and output collection.

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "file_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 size)
  Entry point for the manager process.

5.22.1 Detailed Description

Orchestrates the classification by managing worker coordination and output collection.

Author

Wiktor Szewczyk

Definition in file manager.c.

5.22.2 Function Documentation

5.22.2.1 manager()

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

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

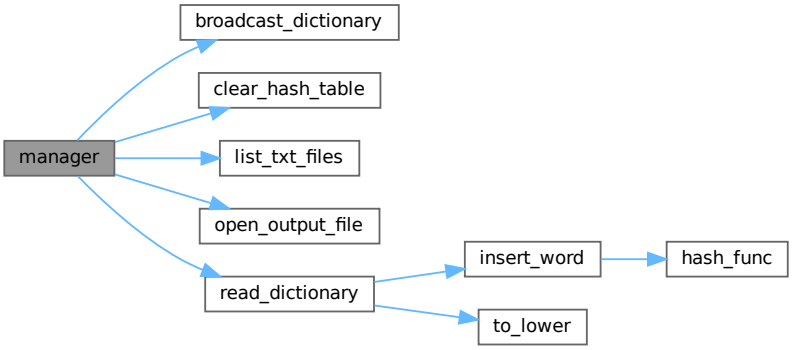
input_dir	Path to directory containing .txt files
dict_file	Path to the dictionary file
output_file	Path where results will be written
size	Number of MPI processes

Definition at line 17 of file manager.c.

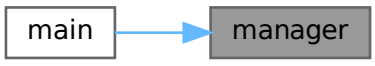
References broadcast_dictionary(), clear_hash_table(), DONE_MSG, FILE_MSG, list_txt_files(), MAX_FILES, MAX_KEYWORDS, open_output_file(), read_dictionary(), REQUEST_MSG, and VEC_MSG.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.23 manager.c

Go to the documentation of this file.

```
00001
00007 #include <mpi.h>
00008 #include <stdio.h>
00009 #include <stdlib.h>
00010 #include <string.h>
00011
00012 #include "file_utils.h"
```

```
00013 #include "hash_table.h"
00014 #include "manager.h"
00015 #include "msg_consts.h"
00016
00017 void manager(const char *input_dir, const char *dict_file, const char *output_file,
00018              int size)
00019 {
00020     char *keywords[MAX_KEYWORDS] = {0};
00021     int num_keywords = read_dictionary(dict_file, keywords, MAX_KEYWORDS);
00022     if (num_keywords < 0)
00023     {
00024         perror("Failed to read dictionary");
00025         MPI_Abort(MPI_COMM_WORLD, 1);
00026     }
00027
00028     broadcast_dictionary(keywords, num_keywords);
00029
00030     char files[MAX_FILES][PATH_MAX] = {0};
00031     int file_count = list_txt_files(input_dir, files, MAX_FILES);
00032     if (file_count < 0)
00033     {
00034         fprintf(stderr, "Error reading directory: %s\n", input_dir);
00035         MPI_Abort(MPI_COMM_WORLD, 1);
00036     }
00037
00038     FILE *out = open_output_file(output_file);
00039     if (!out)
00040     {
00041         perror("Failed to open output file");
00042         MPI_Abort(MPI_COMM_WORLD, 1);
00043     }
00044
00045     fprintf(out, "%-12s:", "dictionary");
00046     for (int i = 0; i < num_keywords; i++)
00047         fprintf(out, " %s", keywords[i]);
00048     fprintf(out, "\n");
00049
00050     int current_file = 0;
00051     int done_workers = 0;
00052
00053     MPI_Request req;
00054     MPI_Status status;
00055     MPI_Irecv(NULL, 0, MPI_CHAR, MPI_ANY_SOURCE, REQUEST_MSG, MPI_COMM_WORLD, &req);
00056
00057     while (done_workers < size - 1)
00058     {
00059         int flag = 0;
00060         MPI_Test(&req, &flag, &status);
00061         if (!flag)
00062             continue;
00063
00064         int source = status.MPI_SOURCE;
00065
00066         if (current_file < file_count)
00067         {
00068             MPI_Send(files[current_file++], PATH_MAX, MPI_CHAR, source, FILE_MSG,
00069                      MPI_COMM_WORLD);
00070         }
00071         else
00072         {
00073             MPI_Send(NULL, 0, MPI_CHAR, source, DONE_MSG, MPI_COMM_WORLD);
00074             done_workers++;
00075             MPI_Irecv(NULL, 0, MPI_CHAR, MPI_ANY_SOURCE, REQUEST_MSG, MPI_COMM_WORLD,
00076                       &req);
00077             continue;
00078         }
00079
00080         int vec[MAX_KEYWORDS];
00081         char fname[PATH_MAX];
00082         MPI_Recv(vec, num_keywords, MPI_INT, source, VEC_MSG, MPI_COMM_WORLD,
00083                  MPI_STATUS_IGNORE);
00084         MPI_Recv(fname, PATH_MAX, MPI_CHAR, source, FILE_MSG, MPI_COMM_WORLD,
00085                  MPI_STATUS_IGNORE);
00086
00087         fprintf(out, "%-12s:", strrchr(fname, '/') ? strrchr(fname, '/') + 1 : fname);
00088         for (int i = 0; i < num_keywords; i++)
00089             fprintf(out, " %d", vec[i]);
00090         fprintf(out, "\n");
00091
00092         MPI_Irecv(NULL, 0, MPI_CHAR, MPI_ANY_SOURCE, REQUEST_MSG, MPI_COMM_WORLD, &req);
00093     }
00094
00095     int flag = 0;
00096     MPI_Test(&req, &flag, MPI_STATUS_IGNORE);
00097     if (!flag)
00098     {
00099         MPI_Cancel(&req);
```

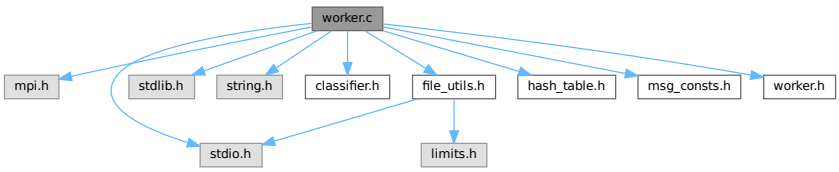
```
00100         MPI_Wait(&req, MPI_STATUS_IGNORE);
00101     }
00102
00103     fclose(out);
00104     for (int i = 0; i < num_keywords; i++)
00105         free(keywords[i]);
00106     clear_hash_table();
00107 }
```

5.24 worker.c File Reference

Logic for a worker process participating in distributed document classification.

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "classifier.h"
#include "file_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 MPI worker process.

5.24.1 Detailed Description

Logic for a worker process participating in distributed document classification.

Author

Wiktor Szewczyk

Definition in file worker.c.

5.24.2 Function Documentation

5.24.2.1 worker()

```
void worker (
            void  )
```

Entry point for each MPI worker process.

Each worker:

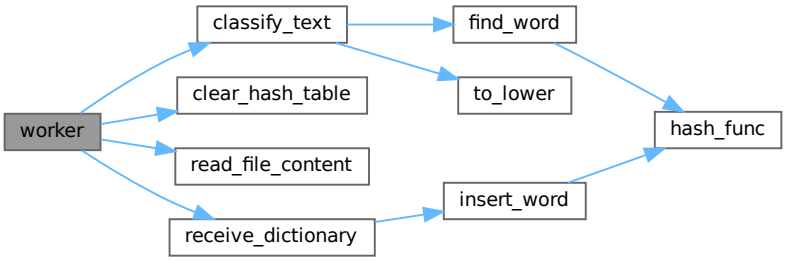
- Receives the dictionary from the manager
- Processes assigned documents
- Sends back feature vectors to the manager

Definition at line 19 of file worker.c.

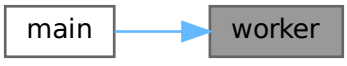
References classify_text(), clear_hash_table(), DONE_MSG, FILE_MSG, MAX_DOC_SIZE, MAX_KEYWORDS, read_file_content(), receive_dictionary(), REQUEST_MSG, and VEC_MSG.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



5.25 worker.c

Go to the documentation of this file.

```
00001
00008 #include <mpi.h>
00009 #include <stdio.h>
00010 #include <stdlib.h>
00011 #include <string.h>
00012
00013 #include "classifier.h"
00014 #include "file_utils.h"
00015 #include "hash_table.h"
00016 #include "msg_consts.h"
00017 #include "worker.h"
00018
00019 void worker(void)
00020 {
00021     char *keywords[MAX_KEYWORDS] = {0};
00022     int num_keywords;
00023     receive_dictionary(keywords, &num_keywords);
00024
00025     int vec[MAX_KEYWORDS] = {0};
00026     char fname[PATH_MAX] = {0};
00027     char buffer[MAX_DOC_SIZE] = {0};
00028
00029     while (1)
00030     {
00031         MPI_Send(NULL, 0, MPI_CHAR, 0, REQUEST_MSG, MPI_COMM_WORLD);
00032
00033         MPI_Status status;
00034         MPI_Probe(0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
00035
00036         if (status.MPI_TAG == DONE_MSG)
00037         {
00038             MPI_Recv(NULL, 0, MPI_CHAR, 0, DONE_MSG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
00039             break;
00040         }
00041
00042         MPI_Recv(fname, PATH_MAX, MPI_CHAR, 0, FILE_MSG, MPI_COMM_WORLD,
00043                  MPI_STATUS_IGNORE);
00044
00045         if (read_file_content(fname, buffer, sizeof(buffer)) < 0)
00046         {
00047             memset(vec, 0, sizeof(vec));
00048         }
00049         else
00050         {
00051             classify_text(buffer, vec, num_keywords);
00052         }
00053
00054         MPI_Request reqs[2];
00055         MPI_Isend(vec, num_keywords, MPI_INT, 0, VEC_MSG, MPI_COMM_WORLD, &reqs[0]);
00056         MPI_Isend(fname, PATH_MAX, MPI_CHAR, 0, FILE_MSG, MPI_COMM_WORLD, &reqs[1]);
00057         MPI_Waitall(2, reqs, MPI_STATUSES_IGNORE);
00058     }
00059
00060     for (int i = 0; i < num_keywords; i++)
00061         free(keywords[i]);
00062     clear_hash_table();
00063 }
```

Index