warsztat4

Generated by Doxygen 1.9.3

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 measurments.c File Reference

Functions

- int main (void)

  Główna funkcja, w której są wykonywane pomiary czasu działania poszczególnych implementacji funkcji memset.

2.1.1 Function Documentation

2.1.1.1 main()

```
int main (
            void  )
```

Główna funkcja, w której są wykonywane pomiary czasu działania poszczególnych implementacji funkcji memset.

Returns

int

Definition at line 29 of file measurments.c.

```
00029                 {
00030 
00031     char buffer1[500], buffer2[1024*1024];
00032 
00033     printf("################# \n");
00034     measure_function(memset(&buffer1[0], '$', sizeof(buffer1)), "memset");
00035     measure_function(memset(&buffer2[0], '$', sizeof(buffer2)), "memset");
00036     printf("################# \n");
00037 
00038     printf("\n");
00039 
00040     memset(&buffer1[0], 0, sizeof(buffer1));
00041     memset(&buffer2[0], 0, sizeof(buffer2));
00042 
00043     printf("################# \n");
00044     measure_function(my_memset_ver1(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver1");
00045     measure_function(my_memset_ver1(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver1");
00046     printf("################# \n");
```

```

    printf("\n");

    memset(&buffer1[0], 0, sizeof(buffer1));
    memset(&buffer2[0], 0, sizeof(buffer2));

    printf("################## \n");
    measure_function(my_memset_ver2(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver2");
    measure_function(my_memset_ver2(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver2");
    printf("################## \n");

    printf("\n");

    memset(&buffer1[0], 0, sizeof(buffer1));
    memset(&buffer2[0], 0, sizeof(buffer2));

    printf("################## \n");
    measure_function(my_memset_ver3(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver3");
    measure_function(my_memset_ver3(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver3");
    printf("################## \n");

    printf("\n");

    memset(&buffer1[0], 0, sizeof(buffer1));
    memset(&buffer2[0], 0, sizeof(buffer2));

    printf("################## \n");
    measure_function(my_memset_ver4(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver4");
    measure_function(my_memset_ver4(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver4");
    printf("################## \n");

    printf("\n");

    memset(&buffer1[0], 0, sizeof(buffer1));
    memset(&buffer2[0], 0, sizeof(buffer2));

    printf("################## \n");
    measure_function(my_memset_ver5(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver5");
    measure_function(my_memset_ver5(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver5");
    printf("################## \n");

    printf("\n");

    printf("################## \n");
    measure_function(my_memset_ver6(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver6");
    measure_function(my_memset_ver6(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver6");
    printf("################## \n");

    printf("\n");

    return 0;
}
```

2.2 measurments.c

Go to the documentation of this file.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "my_memset.h"

# define measure_function( function , label ) \
do { \
struct timespec start = {0}; \
struct timespec end = {0}; \
( void ) clock_gettime ( CLOCK_MONOTONIC , &start); \
( void ) function ; \
( void ) clock_gettime ( CLOCK_MONOTONIC , &end); \
const double seconds = end.tv_sec - start.tv_sec; \
const double nanoseconds = end.tv_nsec - start.tv_nsec; \
const double final_time = \
(( seconds * 1e9 ) + nanoseconds ) * 1e-9; \
printf ( " Measured time of function %s = %lf \n " , \
label , final_time ) ; \
} while (0)

int main(void){

    char buffer1[500], buffer2[1024*1024];

    printf("################## \n");
    measure_function(memset(&buffer1[0], '$', sizeof(buffer1)), "memset");
    measure_function(memset(&buffer2[0], '$', sizeof(buffer2)), "memset");
    printf("################## \n");

    printf("\n");

    memset(&buffer1[0], 0, sizeof(buffer1));
    memset(&buffer2[0], 0, sizeof(buffer2));

    printf("################## \n");
    measure_function(my_memset_ver1(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver1");
    measure_function(my_memset_ver1(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver1");
    printf("################## \n");

    printf("\n");

    memset(&buffer1[0], 0, sizeof(buffer1));
    memset(&buffer2[0], 0, sizeof(buffer2));

    printf("################## \n");
    measure_function(my_memset_ver2(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver2");
    measure_function(my_memset_ver2(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver2");
    printf("################## \n");

    printf("\n");

    memset(&buffer1[0], 0, sizeof(buffer1));
    memset(&buffer2[0], 0, sizeof(buffer2));

    printf("################## \n");
    measure_function(my_memset_ver3(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver3");
    measure_function(my_memset_ver3(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver3");
    printf("################## \n");

    printf("\n");

    memset(&buffer1[0], 0, sizeof(buffer1));
    memset(&buffer2[0], 0, sizeof(buffer2));

    printf("################## \n");
    measure_function(my_memset_ver4(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver4");
    measure_function(my_memset_ver4(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver4");
    printf("################## \n");

    printf("\n");

    memset(&buffer1[0], 0, sizeof(buffer1));
    memset(&buffer2[0], 0, sizeof(buffer2));

    printf("################## \n");
    measure_function(my_memset_ver5(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver5");
    measure_function(my_memset_ver5(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver5");
    printf("################## \n");

    printf("\n");

    printf("################## \n");
    measure_function(my_memset_ver6(&buffer1[0], '$', sizeof(buffer1)), "my_memset_ver6");
    measure_function(my_memset_ver6(&buffer2[0], '$', sizeof(buffer2)), "my_memset_ver6");
    printf("################## \n");

    printf("\n");

    return 0;
}
```

2.3 my_memset.c File Reference

Functions

- void ∗ my_memset_ver1 (void ∗ptr, int value, size_t num)

  Funkcja my_memset_ver1 - pierwsza wersja, czyli ustawianie każdego bajtu na zadaną wartość po kolei (bajt po bajcie).

- void ∗ my_memset_ver2 (void ∗ptr, int value, size_t num)

  Funkcja memset (wersja 2), w której wykorzystano instrukcje wektorowe (kopiuje 32 bajty do tablicy naraz).

- void ∗ my_memset_ver3 (void ∗ptr, int value, size_t num)

    Funkcja memset (wersja 3) z rozwiniętą pętlą (możliwe zwiększenie efektywności kodu).
- void ∗ my_memset_ver4 (void ∗restrict ptr, int value, size_t num)

    Funkcja memset (wersja 4), w której wykorzystano instrukcje wektorowe (kopiuje 32 bajty to tablicy naraz) oraz słowo kluczowe restrict (możliwość optymalizacji przez kompilator).
- void ∗ my_memset_ver5 (void ∗ptr, int value, size_t num)

    Funkcja memset (wersja 5) - kopiuje po 64 bajty do tablicy w jednym przebiegu pętli (za pomocą instrukcji wektorowych).
- void ∗ my_memset_ver6 (void ∗ptr, int value, size_t num)

    Funkcja memset (wersja 6) - kopiuje po 128 bajtów do tablicy w jednym przebiegu pętli (za pomocą instrukcji wektorowych).

2.3.1 Function Documentation

2.3.1.1 my_memset_ver1()

```
void * my_memset_ver1 (
            void * ptr,
            int value,
            size_t num )
```

Funkcja my_memset_ver1 - pierwsza wersja, czyli ustawianie każdego bajtu na zadaną wartość po kolei (bajt po bajcie).

Parameters

in	*ptr*	
in	*value*	
in	*num*	

Returns

void∗

Definition at line 15 of file my_memset.c.

```
                                                           {

    if(ptr == NULL){
        return NULL;
    }

    unsigned char* cptr = (unsigned char*)ptr;

    for(size_t i = 0 ; i < num ; ++i){

        (*cptr) = (unsigned char)value;

        cptr += 1;

    }

    return ptr;

}
```

2.3.1.2 my_memset_ver2()

```
void * my_memset_ver2 (
            void * ptr,
            int value,
            size_t num )
```

Funkcja memset (wersja 2), w której wykorzystano instrukcje wektorowe (kopiuje 32 bajty do tablicy naraz).

Parameters

in	*ptr*	
in	*value*	
in	*num*	

Returns

void*

Definition at line 43 of file my_memset.c.

```
                                                   {

    if(ptr == NULL){
        return NULL;
    }

    __m256i v = _mm256_set1_epi8((unsigned char)value);

    size_t number_of_iterations = num/sizeof(__m256i);
    size_t rest = num%sizeof(__m256i);

    for(size_t i = 0 ; i < number_of_iterations ; ++i){

        _mm256_storeu_si256((__m256i*)ptr, v);

        ptr += sizeof(__m256i);

    }

    unsigned char* cptr2 = (unsigned char*)ptr;

    for(size_t i = 0 ; i < rest ; ++i){

        (*cptr2) = (unsigned char)value;

        cptr2 += 1;

    }

    return ptr;

}
```

2.3.1.3 my_memset_ver3()

```
void * my_memset_ver3 (
            void * ptr,
            int value,
            size_t num )
```

Funkcja memset (wersja 3) z rozwiniętą pętlą (możliwe zwiększenie efektywności kodu).

Parameters

in	*ptr*	
in	*value*	
in	*num*	

Returns

void∗

Definition at line 84 of file my_memset.c.

```
                                                            {
    if(ptr == NULL){
        return NULL;
    }

    unsigned char* _ptr = (unsigned char*)ptr;

    unsigned char casted_value = (unsigned char)value;

    size_t batch_size = 16;

    size_t number_of_iterations = num/batch_size;
    size_t rest = num%batch_size;

    size_t p = 0;

    for(size_t i = 0 ; i < number_of_iterations ; i++){

        _ptr[p] = casted_value;
        _ptr[p + 1] = casted_value;
        _ptr[p + 2] = casted_value;
        _ptr[p + 3] = casted_value;
        _ptr[p + 4] = casted_value;
        _ptr[p + 5] = casted_value;
        _ptr[p + 6] = casted_value;
        _ptr[p + 7] = casted_value;
        _ptr[p + 8] = casted_value;
        _ptr[p + 9] = casted_value;
        _ptr[p + 10] = casted_value;
        _ptr[p + 11] = casted_value;
        _ptr[p + 12] = casted_value;
        _ptr[p + 13] = casted_value;
        _ptr[p + 14] = casted_value;
        _ptr[p + 15] = casted_value;

        p += 16;

    }

    _ptr += number_of_iterations*16;

    for(size_t i = 0 ; i < rest ; ++i){

        _ptr[i] = casted_value;

    }

    return ptr;
}
```

2.3.1.4 my_memset_ver4()

```
void * my_memset_ver4 (
            void *restrict ptr,
            int value,
            size_t num )
```

Funkcja memset (wersja 4), w której wykorzystano instrukcje wektorowe (kopiuje 32 bajty to tablicy naraz) oraz słowo kluczowe restrict (możliwość optymalizacji przez kompilator).

Parameters

in	*ptr*	
in	*value*	
in	*num*	

Returns

void∗

Definition at line 144 of file my_memset.c.

```
{

    if(ptr == NULL){
        return NULL;
    }

    __m256i v = _mm256_set1_epi8((unsigned char)value);

    size_t number_of_iterations = num/sizeof(__m256i);
    size_t rest = num%sizeof(__m256i);

    for(size_t i = 0 ; i < number_of_iterations ; ++i){

        _mm256_storeu_si256((__m256i*)ptr, v);

        ptr += sizeof(__m256i);

    }

    unsigned char* cptr2 = (unsigned char*)ptr;

    for(size_t i = 0 ; i < rest ; ++i){

        (*cptr2) = (unsigned char)value;

        cptr2 += 1;

    }

    return ptr;

}
```

2.3.1.5 my_memset_ver5()

```
void * my_memset_ver5 (
            void * ptr,
            int value,
            size_t num )
```

Funkcja memset (wersja 5) - kopiuje po 64 bajty do tablicy w jednym przebiegu pętli (za pomocą instrukcji wektorowych).

Parameters

in	*ptr*	
in	*value*	
in	*num*	

Returns

void∗

Definition at line 186 of file my_memset.c.

```
                                                         {

    if(ptr == NULL){
        return NULL;
    }

    __m256i v1 = _mm256_set1_epi8((unsigned char) value);

    size_t number_of_iterations = (num)/(2*sizeof(__m256i));
    size_t rest = (num)%(2*sizeof(__m256i));

    for(size_t i = 0 ; i < number_of_iterations ; i++){

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

    }

    unsigned char* cptr = (unsigned char*)ptr;

    for(size_t i = 0 ; i < rest ; i++){
        *cptr = (unsigned char)value;

        cptr += 1;
    }

    return ptr;

}
```

2.3.1.6 my_memset_ver6()

```
void * my_memset_ver6 (
            void * ptr,
            int value,
            size_t num )
```

Funkcja memset (wersja 6) - kopiuje po 128 bajtów do tablicy w jednym przebiegu pętli (za pomocą instrukcji wektorowych).

Parameters

in	*ptr*	
in	*value*	
in	*num*	

Returns

void∗

Definition at line 230 of file my_memset.c.

```
                                                         {

```

```
    if(ptr == NULL){
        return NULL;
    }

    __m256i v1 = _mm256_set1_epi8((unsigned char) value);

    size_t number_of_iterations = (num)/(4*sizeof(__m256i));
    size_t rest = (num)%(4*sizeof(__m256i));

    for(size_t i = 0 ; i < number_of_iterations ; i++){

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

    }

    unsigned char* cptr = (unsigned char*)ptr;

    for(size_t i = 0 ; i < rest ; i++){
        *cptr = (unsigned char)value;

        cptr += 1;
    }

    return ptr;

}
```

2.4 my_memset.c

Go to the documentation of this file.

```

#include <stdio.h>
#include <immintrin.h>
#include <stdint.h>
#include <string.h>

void* my_memset_ver1(void* ptr, int value, size_t num){

    if(ptr == NULL){
        return NULL;
    }

    unsigned char* cptr = (unsigned char*)ptr;

    for(size_t i = 0 ; i < num ; ++i){

         (*cptr) = (unsigned char)value;

        cptr += 1;

    }

    return ptr;

}

void* my_memset_ver2(void* ptr, int value, size_t num){

    if(ptr == NULL){
        return NULL;
    }

    __m256i v = _mm256_set1_epi8((unsigned char)value);

    size_t number_of_iterations = num/sizeof(__m256i);
    size_t rest = num%sizeof(__m256i);

```

```c
    for(size_t i = 0 ; i < number_of_iterations ; ++i){

        _mm256_storeu_si256((__m256i*)ptr, v);

        ptr += sizeof(__m256i);

    }

    unsigned char* cptr2 = (unsigned char*)ptr;

    for(size_t i = 0 ; i < rest ; ++i){

        (*cptr2) = (unsigned char)value;

        cptr2 += 1;

    }

    return ptr;
}

void* my_memset_ver3(void* ptr, int value, size_t num){

    if(ptr == NULL){
        return NULL;
    }

    unsigned char* _ptr = (unsigned char*)ptr;

    unsigned char casted_value = (unsigned char)value;

    size_t batch_size = 16;

    size_t number_of_iterations = num/batch_size;
    size_t rest = num%batch_size;

    size_t p = 0;

    for(size_t i = 0 ; i < number_of_iterations ; i++){

        _ptr[p] = casted_value;
        _ptr[p + 1] = casted_value;
        _ptr[p + 2] = casted_value;
        _ptr[p + 3] = casted_value;
        _ptr[p + 4] = casted_value;
        _ptr[p + 5] = casted_value;
        _ptr[p + 6] = casted_value;
        _ptr[p + 7] = casted_value;
        _ptr[p + 8] = casted_value;
        _ptr[p + 9] = casted_value;
        _ptr[p + 10] = casted_value;
        _ptr[p + 11] = casted_value;
        _ptr[p + 12] = casted_value;
        _ptr[p + 13] = casted_value;
        _ptr[p + 14] = casted_value;
        _ptr[p + 15] = casted_value;

        p += 16;

    }

    _ptr += number_of_iterations*16;

    for(size_t i = 0 ; i < rest ; ++i){

        _ptr[i] = casted_value;

    }

    return ptr;
}

void* my_memset_ver4(void* restrict ptr, int value, size_t num){

    if(ptr == NULL){
        return NULL;
    }

    __m256i v = _mm256_set1_epi8((unsigned char)value);

    size_t number_of_iterations = num/sizeof(__m256i);
    size_t rest = num%sizeof(__m256i);

    for(size_t i = 0 ; i < number_of_iterations ; ++i){

        _mm256_storeu_si256((__m256i*)ptr, v);

        ptr += sizeof(__m256i);

    }

    unsigned char* cptr2 = (unsigned char*)ptr;

    for(size_t i = 0 ; i < rest ; ++i){

        (*cptr2) = (unsigned char)value;

        cptr2 += 1;

    }

    return ptr;

}

void* my_memset_ver5(void* ptr, int value, size_t num){

    if(ptr == NULL){
        return NULL;
    }

    __m256i v1 = _mm256_set1_epi8((unsigned char) value);

    size_t number_of_iterations = (num)/(2*sizeof(__m256i));
    size_t rest = (num)%(2*sizeof(__m256i));

    for(size_t i = 0 ; i < number_of_iterations ; i++){

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

    }

    unsigned char* cptr = (unsigned char*)ptr;

    for(size_t i = 0 ; i < rest ; i++){
        *cptr = (unsigned char)value;

        cptr += 1;
    }

    return ptr;

}

void* my_memset_ver6(void* ptr, int value, size_t num){

    if(ptr == NULL){
        return NULL;
    }

    __m256i v1 = _mm256_set1_epi8((unsigned char) value);

    size_t number_of_iterations = (num)/(4*sizeof(__m256i));
    size_t rest = (num)%(4*sizeof(__m256i));

    for(size_t i = 0 ; i < number_of_iterations ; i++){

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

        _mm256_storeu_si256((__m256i*)ptr, v1);

        ptr += sizeof(__m256i);

    }

    unsigned char* cptr = (unsigned char*)ptr;

    for(size_t i = 0 ; i < rest ; i++){
        *cptr = (unsigned char)value;

        cptr += 1;
    }

    return ptr;

}

```

2.5 my_memset.h

```
#ifndef MY_MEMSET_H
#define MY_MEMSET_H

void* my_memset_ver1(void* ptr, int value, size_t num);
void* my_memset_ver2(void* ptr, int value, size_t num);
void* my_memset_ver3(void* ptr, int value, size_t num);
void* my_memset_ver4(void* ptr, int value, size_t num);
void* my_memset_ver5(void* ptr, int value, size_t num);
void* my_memset_ver6(void* ptr, int value, size_t num);

#endif
```

Index