This depositary provides the implemetation of Memory pool and Memory storage, both are with high performance. They are all head files, coded with C++11.

User defined data struct(POD):

struct Dem {

char data[1024\*8]; //8k byte

};

[MemoryPool]: the usage is showed below:

MemoryPool<T>(SizeType size\_type, size\_t pool\_size) ;//T: std data, or user-defined pod-type data struct; size\_type can be any num of pieces of T type memory, like 1, 16,etc. Pool\_size is the inited size of the pool.

These are some defines:

MemoryPool<Dem> pool(10, 128);

MemoryPool<int> pool(4, 1024);

MemoryPool<float> pool(64, 128);

Get memorys:

Auto p = pool.Alloc();

Free memorys:

Pool.Free(p);

[MemoryStorage]: the usage is showed below:

MemoryStorage<T>(size\_t init\_size, std::initializer\_list<size\_t> il\_type\_sizes); //T: std data, or user-defined pod-type data struct; init\_size is the inited size of each size-type memory pool. il\_type\_sizes is the size-type init list, like:{2, 8, 16, 1024, 128, 31}.

These are some defines:

MemoryStorage<Dem> storage(10, { 1, 699, 102, 32});

MemoryStorage<int> storage(64, { 102, 32});

MemoryStorage<char> storage(1, { 1});

Get memorys:

Auto p = storage.Alloc(10);

Free memorys:

storage.Free(p);

Test results compared with new-delete and std::vector:

Test platform: cpu: i3, memory: 8G.

Test cases:

test\_new\_delete();

test\_vectorMemPool();

test\_memoryPool();

test\_memoryStorage();

//test new delete

void test\_new\_delete() {

auto start = clock();

for (int j = 0; j < LOOP; j++) {

for (int i = 0; i < MEM\_NUM; i++) {

Dem\* p = new Dem;

delete p;

}

}

std::cout << "\nNew\_delete Time ms: ";

std::cout << (((double)clock() - start)) << std::endl;

}

//test vector memory alloc and deallac.

void test\_vectorMemPool() {

std::vector<Dem> stackVector;

stackVector.reserve(MEM\_NUM);

auto start = clock();

for (int j = 0; j < LOOP; j++) {

assert(stackVector.empty());

for (int i = 0; i < MEM\_NUM; i++)

stackVector.emplace\_back(Dem());

for (int i = 0; i < MEM\_NUM; i++)

stackVector.pop\_back();

}

std::cout << "\nVector Time ms: ";

std::cout << (((double)clock() - start)) << std::endl;

}

//test memory pool

void test\_memoryPool() {

MemoryPool<Dem> pool(1, MEM\_NUM);

auto start = clock();

for (int j = 0; j < LOOP; j++) {

for (int i = 0; i < MEM\_NUM; i++) {

auto p = pool.Alloc();

pool.Free(p);

}

}

std::cout << "\nMemoryPool Time ms: ";

std::cout << (((double)clock() - start)) << std::endl;

}

//test memory storage

void test\_memoryStorage() {

MemoryStorage<Dem> storage(MEM\_NUM, { 1 });

auto start = clock();

for (int j = 0; j < LOOP; j++) {

for (int i = 0; i < MEM\_NUM; i++) {

auto p = storage.Alloc(1);

storage.Free(p);

}

}

std::cout << "\nMemoryStorage Time ms: ";

std::cout << (((double)clock() - start)) << std::endl;

}

1. : with alloc big memory:

Params setting:

// the numbers of wanted pieces of memory.

#define MEM\_NUM 10000

// loops of each memory test and total memory times of alloc-and-dealloc are LOOP \* MEM\_NUM.

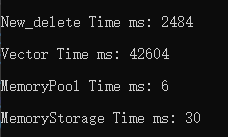
#define LOOP 500

struct Dem {

char data[1024\*64]; //64k byte

};

The time unit is : millisecond(ms).



1. : with alloc small memory:

Params setting:

// the numbers of wanted pieces of memory.

#define MEM\_NUM 10000

// loops of each memory test and total memory times of alloc-and-dealloc are LOOP \* MEM\_NUM.

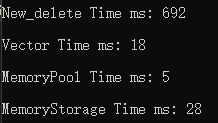
#define LOOP 500

struct Dem {

char data[8]; //8 byte

};

The time unit is : millisecond(ms).



1. : with mediate size of memory:

// the numbers of wanted pieces of memory.

#define MEM\_NUM 10000

// loops of each memory test and total memory times of alloc-and-dealloc are LOOP \* MEM\_NUM.

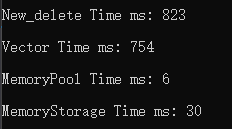
#define LOOP 500

struct Dem {

char data[512]; //512byte

};

The time unit is : millisecond(ms).



Inclusion, the performance of the MemoryPool and MemoryStorage are high and stable, no matter the size of memory allocated , according to the new-delete and std::vector.