성능 측정

Object Free List TLS

노드 1개 크기: 1000Byte

할당받는 노드 수: 1000개

테스트 코드 반복 수: 1,000,000회

최초

(100ns)

Name	Average	Min	Max	Call
alloc	94.5799	83	6170	999998
free	81.7945	68	343	999998

(100ns)

Name	Average	Min	Max	Call
alloc	94.3028	87	5517	999998
free	81.2172	71	888	999998

(100ns)

Name	Average	Min	Max	Call
alloc	94.6592	87	4746	999998
free	81.5574	72	332	999998

(100ns)

Name	Average	Min	Max	Call
new	824.6195	495	7595	999998
delete	307.6109	234	4134	999998

(100ns)

Name	Average	Min	Max	Call
new	829.0779	490	7378	999998
delete	305.2342	224	3732	999998

(100ns)

Name	Average	Min	Max	Call
new	830.5550	485	7332	999998
delete	300.6478	232	4492	999998

개선 1

-- 코드 개선

지역 변수로의 복사를 줄여서 개선

```
    int allocNodeIdx = --chunk->_leftNodeCnt;
00007FF7AB8D27E4  mov         rax,qword ptr [chunk]
00007FF7AB8D27E9  mov         eax,dword ptr [rax+1Ch]
00007FF7AB8D27EC  dec         eax
00007FF7AB8D27EE  mov         dword ptr [rsp+20h],eax
00007FF7AB8D27F2  mov         rax,qword ptr [chunk]
00007FF7AB8D27F7  mov         ecx,dword ptr [rsp+20h]
00007FF7AB8D27FB  mov         dword ptr [rax+1Ch],ecx
00007FF7AB8D27FE  mov         eax,dword ptr [rsp+20h]
00007FF7AB8D2802  mov         dword ptr [allocNodeIdx],eax
```

```
    chunk->_leftFreeCnt -= 1;
00007FF6AD7827E4  mov         rax,qword ptr [chunk]
00007FF6AD7827E9  mov         eax,dword ptr [rax+20h]
00007FF6AD7827EC  dec         eax
00007FF6AD7827EE  mov         rcx,qword ptr [chunk]
00007FF6AD7827F3  mov         dword ptr [rcx+20h],eax

    int allocNodeIdx = chunk->_leftNodeCnt;
00007FF6AD7827F6  mov         rax,qword ptr [chunk]
00007FF6AD7827FB  mov         eax,dword ptr [rax+1Ch]
00007FF6AD7827FE  mov         dword ptr [allocNodeIdx],eax
```

```
    stAllocTlsNode<T>* allocNode = &chunk->_nodes[allocNodeIdx];
00007FF600EF2802  movsxd      rax,dword ptr [allocNodeIdx]
00007FF600EF2807  imul        rax,rax,400h
00007FF600EF280E  mov         rcx,qword ptr [chunk]
00007FF600EF2813  add         rax,qword ptr [rcx+8]
00007FF600EF2817  mov         qword ptr [allocNode],rax
    T* allocData = &allocNode->_data;
00007FF600EF281C  mov         rax,qword ptr [allocNode]
00007FF600EF2821  add         rax,8
00007FF600EF2825  mov         qword ptr [allocData],rax
```

```
    T* allocData = &chunk->_nodes[allocNodeIdx]._data;
00007FF6AD782802  movsxd      rax,dword ptr [allocNodeIdx]
00007FF6AD782807  imul        rax,rax,400h
00007FF6AD78280E  mov         rcx,qword ptr [chunk]
00007FF6AD782813  mov         rcx,qword ptr [rcx+8]
00007FF6AD782817  lea         rax,[rcx+rax+8]
00007FF6AD78281C  mov         qword ptr [allocData],rax
```

```
    chunk->_leftFreeCnt -= 1;
00007FF7866528B8  mov         rax,qword ptr [chunk]
00007FF7866528BD  mov         eax,dword ptr [rax+20h]
00007FF7866528C0  dec         eax
00007FF7866528C2  mov         rcx,qword ptr [chunk]
00007FF7866528C7  mov         dword ptr [rcx+20h],eax

    if(chunk->_leftFreeCnt == 0){
00007FF7866528CA  mov         rax,qword ptr [chunk]
00007FF7866528CF  cmp         dword ptr [rax+20h],0
00007FF7866528D3  jne         CObjectFreeListTLS<stNode>::freeObject+6Bh (07FF7866528EBh)
```

```
    if(--chunk->_leftFreeCnt == 0){
00007FF72A4728B8  mov         rax,qword ptr [chunk]
00007FF72A4728BD  mov         eax,dword ptr [rax+20h]
00007FF72A4728C0  dec         eax
00007FF72A4728C2  mov         dword ptr [rsp+20h],eax
00007FF72A4728C6  mov         rax,qword ptr [chunk]
00007FF72A4728CB  mov         ecx,dword ptr [rsp+20h]
00007FF72A4728CF  mov         dword ptr [rax+20h],ecx
00007FF72A4728D2  cmp         dword ptr [rsp+20h],0
00007FF72A4728D7  jne         CObjectFreeListTLS<stNode>::freeObject+6Fh (07FF72A4728EFh)
```

한줄로 처리하지 않고 2줄로 처리하면

Mov 명령이 하나 줄어든다.

```
    stAllocTlsNode<T>* node = ((stAllocTlsNode<T>*)((unsigned __int64)object + _dataToNodePtr));
00007FF6A9D7289A  mov         rax,qword ptr [this]
00007FF6A9D7289F  mov         rax,qword ptr [rax+400030h]
00007FF6A9D728A6  mov         rcx,qword ptr [object]
00007FF6A9D728AB  add         rcx,rax
00007FF6A9D728AE  mov         rax,rcx
00007FF6A9D728B1  mov         qword ptr [node],rax
    stAllocChunk<T>* chunk = node->_afflicatedChunk;
00007FF6A9D728B6  mov         rax,qword ptr [node]
00007FF6A9D728BB  mov         rax,qword ptr [rax+3F8h]
00007FF6A9D728C2  mov         qword ptr [chunk],rax
```

```
    stAllocChunk<T>* chunk = ((stAllocTlsNode<T>*)((unsigned __int64)object + _dataToNodePtr))->_afflicatedCh
00007FF78665289A  mov         rax,qword ptr [this]
00007FF78665289F  mov         rax,qword ptr [rax+400030h]
00007FF7866528A6  mov         rcx,qword ptr [object]
00007FF7866528AB  mov         rax,qword ptr [rcx+rax+3F8h]
00007FF7866528B3  mov         qword ptr [chunk],rax
```

위가 더 빠른데 이유를 모르겠다.

(100ns)

Name	Average	Min	Max	Call
alloc	88.0189	77	6444	999998
free	82.2557	67	352	999998

(100ns)

Name	Average	Min	Max	Call
alloc	87.7829	81	5502	999998
free	82.2803	71	323	999998

(100ns)

Name	Average	Min	Max	Call
alloc	87.9238	80	4585	999998
free	82.0696	71	300	999998

개선 2

- 변화가 없는 변수를 constexpr 키워드 적용해서 코드에 상수로 들어가도록 수정

- 기존에는 _dataToNodePtr을 생성자에서 계산해서 사용했었음
- 그러다보니 계산할때마다 변수에서 로드하는 코드가 생성됨
- 해당 변수를 constexpr로 변경하여 수동으로 초기화하도록한다.
- 대신 생성자에서는 계산을 통해 변수 값이 정상인지를 평가한다.

```
    stAllocTlsNode<T>* node = ((stAllocTlsNode<T>*)((unsigned __int64)object + _
00007FF6A9D7289A  mov         rax,qword ptr [this]
00007FF6A9D7289F  mov         rax,qword ptr [rax+400030h]
00007FF6A9D728A6  mov         rcx,qword ptr [object]
00007FF6A9D728AB  add         rcx,rax
00007FF6A9D728AE  mov         rax,rcx
00007FF6A9D728B1  mov         qword ptr [node],rax
    stAllocChunk<T>* chunk = node->_afflicatedChunk;
00007FF6A9D728B6  mov         rax,qword ptr [node]
00007FF6A9D728BB  mov         rax,qword ptr [rax+3F8h]
00007FF6A9D728C2  mov         qword ptr [chunk],rax
```

```
    stAllocChunk<T>* chunk = ((stAllocTlsNode<T>*)((unsigned __int64)object + _dataToNodePtr))->_afflicatedChu
00007FF74E1F288A  mov         rax,qword ptr [object]
00007FF74E1F288F  mov         rax,qword ptr [rax+3F0h]
00007FF74E1F2896  mov         qword ptr [chunk],rax
```

```
    stAllocTlsNode<T>* node = ((stAllocTlsNode<T>*)((unsigned __int64)object + _dataToNodePtr));
00007FF70FDB288A  mov         rax,qword ptr [object]
00007FF70FDB288F  sub         rax,8
00007FF70FDB2893  mov         qword ptr [node],rax
    stAllocChunk<T>* chunk = node->_afflicatedChunk;
00007FF70FDB2898  mov         rax,qword ptr [node]
00007FF70FDB289D  mov         rax,qword ptr [rax+3F8h]
00007FF70FDB28A4  mov         qword ptr [chunk],rax
```

개선2 결과

(100ns)

Name	Average	Min	Max	Call
alloc	88.7902	82	5844	999998
free	77.2833	67	347	999998

(100ns)

Name	Average	Min	Max	Call
alloc	89.3116	75	5416	999998
free	77.2731	62	327	999998

(100ns)

Name	Average	Min	Max	Call
alloc	88.5043	81	4798	999998
free	77.0870	65	312	999998

개선 3

Chunk에서 node를 가져올 때,
node 배열의 index로
접근해서 가져오고 있다.

```
    chunk->_leftNodeCnt -= 1;
00007FF78D192514  mov         rax,qword ptr [chunk]
00007FF78D192519  mov         eax,dword ptr [rax+1Ch]
00007FF78D19251C  dec         eax
00007FF78D19251E  mov         rcx,qword ptr [chunk]
00007FF78D192523  mov         dword ptr [rcx+1Ch],eax
    int allocNodeIdx = chunk->_leftNodeCnt;
00007FF78D192526  mov         rax,qword ptr [chunk]
00007FF78D19252B  mov         eax,dword ptr [rax+1Ch]
00007FF78D19252E  mov         dword ptr [allocNodeIdx],eax

    T* allocData = &chunk->_nodes[allocNodeIdx]._data;
00007FF78D192532  movsxd      rax,dword ptr [allocNodeIdx]
00007FF78D192537  imul        rax,rax,400h
00007FF78D19253E  mov         rcx,qword ptr [chunk]
00007FF78D192543  mov         rcx,qword ptr [rcx+8]
00007FF78D192547  lea         rax,[rcx+rax+8]
00007FF78D19254C  mov         qword ptr [allocData],rax
```

Chunk에 할당할 node의 pointer를 갖고, 원하면 pointer를 얻을 수 있게 하면 간결한 코드가 나올 것이다.

```
    T* allocData = &chunk->_allocNode++->_data;
00007FF7F6562514  mov         rax,qword ptr [chunk]
00007FF7F6562519  mov         rax,qword ptr [rax+18h]
00007FF7F656251D  add         rax,8
00007FF7F6562521  mov         qword ptr [allocData],rax
00007FF7F6562526  mov         rax,qword ptr [chunk]
00007FF7F656252B  mov         rax,qword ptr [rax+18h]
00007FF7F656252F  add         rax,400h
00007FF7F6562535  mov         rcx,qword ptr [chunk]
00007FF7F656253A  mov         qword ptr [rcx+18h],rax
```

개선 결과

(100ns)

Name	Average	Min	Max	Call
alloc	83.4161	76	5841	999998
free	74.9230	63	395	999998

(100ns)

Name	Average	Min	Max	Call
alloc	83.5443	76	5208	999998
free	74.6061	63	372	999998

(100ns)

Name	Average	Min	Max	Call
alloc	84.1005	74	4311	999998
free	74.3754	58	339	999998

개선 4

TLS node에서 underflow, overflow 체크하던 변수 제거

Underflow 변수가 사라지면 T type 변수의 주소가 곧 node의 주소가 되기 때문에 node 접근을 위한 계산이 사라짐

(100ns)

Name	Average	Min	Max	Call
alloc	84.3352	76	6295	999998
free	67.1615	60	1004	999998

(100ns)

Name	Average	Min	Max	Call
alloc	84.6844	71	5839	999998
free	67.6283	53	260	999998

(100ns)

Name	Average	Min	Max	Call
alloc	84.2640	77	5479	999998
free	67.7216	60	406	999998

Thread 1

Node Size: 10Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
new	369.0592	328	2654	99996
delete	201.4510	183	1706	99996

Node Size: 500Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
new	476.3561	417	3080	99996
delete	218.5927	192	1437	99996

Node Size: 1000Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
new	476.4971	391	5662	99996
delete	224.2225	192	2549	99996

Node Size: 10, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
alloc	48.8535	45	887	99996
free	36.6045	34	745	99996

Node Size: 500Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
alloc	47.9208	45	1260	99996
free	36.4105	34	770	99996

Node Size: 1000Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
alloc	49.6744	45	2217	99996
free	37.4044	34	4387	99996

Node Size: 10Byte, Alloc Num Each Thread: 10000

(100ns)

Name	Average	Min	Max	Call
alloc	480.1409	454	3384	99996
free	353.8023	335	2834	99996

Node Size: 500Byte, Alloc Num Each Thread: 10000

(100ns)

Name	Average	Min	Max	Call
alloc	475.9750	451	13671	99996
free	370.6463	346	3002	99996

Node Size: 1000Byte, Alloc Num Each Thread: 10000

(100ns)

Name	Average	Min	Max	Call
alloc	478.1441	450	32057	99996
free	391.8998	361	4116	99996

Node Size: 10Byte, Alloc Num Each Thread: 50000

(100ns)

Name	Average	Min	Max	Call
alloc	2471.2674	2294	9880	99996
free	1832.3384	1696	7819	99996

Node Size: 500Byte, Alloc Num Each Thread: 50000

(100ns)

Name	Average	Min	Max	Call
alloc	2404.9595	2253	68278	99996
free	2074.1327	1907	16960	99996

Node Size: 1000Byte, Alloc Num Each Thread: 50000

(100ns)

Name	Average	Min	Max	Call
alloc	2394.0611	2254	115819	99996
free	2159.4837	1993	17588	99996

Thread 2

Node Size: 10Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
new	336.2132	248	2980	199994
delete	207.1132	170	2341	199994

Node Size: 500Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
new	518.6033	302	7694	199994
delete	244.4535	181	4087	199994

Node Size: 1000Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
new	585.3017	340	8875	199994
delete	243.7402	178	4634	199994

Node Size: 10Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
alloc	62.9216	46	1323	199994
free	46.9818	34	1332	199994

Node Size: 500Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
alloc	56.6735	46	3447	199994
free	42.3857	34	1567	199994

Node Size: 1000Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
alloc	56.6813	46	3730	199994
free	43.1625	34	905	199994

Node Size: 10Byte, Alloc Num Each Thread: 10000

(100ns)

Name	Average	Min	Max	Call
alloc	518.1296	458	3319	199994
free	384.4094	342	3407	199994

Node Size: 500Byte, Alloc Num Each Thread: 10000

(100ns)

Name	Average	Min	Max	Call
alloc	515.8487	461	16555	199994
free	413.1195	368	2558	199994

Node Size: 1000Byte, Alloc Num Each Thread: 10000

(100ns)

Name	Average	Min	Max	Call
alloc	516.6268	456	54263	199994
free	445.2095	377	3345	199994

Node Size: 10Byte, Alloc Num Each Thread: 50000

(100ns)

Name	Average	Min	Max	Call
alloc	2623.5299	2334	15858	199994
free	1955.8106	1763	13268	199994

Node Size: 500Byte, Alloc Num Each Thread: 50000

(100ns)

Name	Average	Min	Max	Call
alloc	2600.5987	2371	124673	199994
free	2324.1500	2089	19160	199994

Node Size: 1000Byte, Alloc Num Each Thread: 50000

(100ns)

Name	Average	Min	Max	Call
alloc	2708.1428	2300	266782	199994
free	2720.1309	2119	23075	199994

Thread 4

Node Size: 10Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
new	360.3867	251	3877	399990
delete	242.3627	171	3677	399990

Node Size: 500Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
new	703.8732	287	9953	399990
delete	326.0682	180	6265	399990

Node Size: 1000Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
new	789.2717	353	13969	399990
delete	347.7115	183	9944	399990

Node Size: 10Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
alloc	77.2475	48	1487	399990
free	57.0738	35	1376	399990

Node Size: 500Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
alloc	73.7124	46	3727	399990
free	54.6987	35	826	399990

Node Size: 1000Byte, Alloc Num Each Thread: 1000

(100ns)

Name	Average	Min	Max	Call
alloc	78.1311	47	9286	399990
free	58.5199	35	2192	399990

Node Size: 10Byte, Alloc Num Each Thread: 10000

(100ns)

Name	Average	Min	Max	Call
alloc	768.8657	495	5007	399990
free	575.4944	367	3349	399990

Node Size: 500Byte, Alloc Num Each Thread: 10000

(100ns)

Name	Average	Min	Max	Call
alloc	730.9956	499	47133	399990
free	583.2521	409	5859	399990

Node Size: 1000Byte, Alloc Num Each Thread: 10000

(100ns)

Name	Average	Min	Max	Call
alloc	747.2776	491	78735	399990
free	616.5758	422	5357	399990

Node Size: 10Byte, Alloc Num Each Thread: 50000

(100ns)

Name	Average	Min	Max	Call
alloc	3624.1045	2294	24355	399990
free	2695.0308	1728	11427	399990

Node Size: 500Byte, Alloc Num Each Thread: 50000

(100ns)

Name	Average	Min	Max	Call
alloc	3655.4682	2358	222025	399990
free	4167.0450	2025	25599	399990

Node Size: 1000Byte, Alloc Num Each Thread: 50000

(100ns)

Name	Average	Min	Max	Call
alloc	3648.3778	2461	426299	399990
free	5029.3449	2262	34504	399990

개선 5