Ассемблерная оптимизация

НОВИКОВ СЕРГЕЙ

Б05-931

Задачи работы:

- Изучить методы ассемблерной оптимизации программы
- Узнать насколько ускоряется программа после ассемблерных оптимизаций

До оптимизации

Имя функции	% затраченного эксклю...
list::find	33,16%
invoke_main	28,08%
hash_xor	17,32%
__CheckForDebugger...	8,57%
main	7,87%
_RTC_CheckEsp	4,83%
operator new	0,07%
list::append	0,04%
node::node	0,02%
fopen_s	0,01%
fread	0,01%
`vector constructor it...	0,00%
SetUnhandledExcepti...	0,00%
_chkstk	0,00%

Find занимает почти
33,16%
А hash_xor
17,32%
В сумме это почти 50%
поэтому стоит их
оптимизировать.

Оптимизация find

```
char* find(char* str) {
    if (head == NULL)
        return NULL;
    node* this_ = head;

    while (this_ != NULL) {
        if (strcmp(this_->s, str) == 0)
            return this_->s;
        else
            this_ = this_->next;
    }
    return NULL
}
```

Функция до оптимизации.
Перепишем её на ассемблере.

Функция после оптимизации

Вот такая ассемблерная вставка получилась

```
char* find(char* str) {
    char* ret_str = NULL;
    __asm
    {
        mov esi, str
        mov ebx, head

        next_node : cmp ebx, 0
        je notfound
        mov edi, [ebx + 4]
        call my_strcmp
        cmp eax, 0
        je found
        mov ebx, [ebx]
        jmp next_node

        notfound : mov ret_str, 0
        jmp end_nodes
        found : mov ret_str, edi
        end_nodes :
    }
    return ret_str;
}
```

Оптимизация hash_xor

```
int hash_xor(char* str) {
    int result = 0;
    int firstbit = 0;

    int i = 0;
    while (str[i] != 0) {
        result ^= str[i];
        firstbit = (result >> 31) & 1;
        result <<= 1;
        result |= firstbit;
        i++;
    }
    return result;
}
```

Функция до оптимизации.
(Эту функцию перепишем
в отдельном .asm файле)

Hash после оптимизации

Вот код переписанной функции->

А в .cpp файле пишем:

```
extern "C"
{
    int hash_xor(char* str);
}
```

Посмотрим какие
результаты это принесло

```
1   .686
2   .MODEL FLAT, C
3   .CODE
4
5   hash_xor    PROC arg1:ptr byte
6
7               mov eax, 0
8               mov ebx, arg1
9               xor ecx, ecx
10  nextsimb:
11              cmp [ebx], ecx
12              je endstr
13                  xor eax, [ebx]
14                  rol eax, 1
15                  inc ebx
16                  jmp nextsimb
17  endstr:
18              retn
19  hash_xor endp
20
21  END
```

Результаты

	Od	O2	asm
test1	15786	5391	7358
test2	14535	5118	6934
test3	23974	8935	12230
test4	25833	9534	12945
test5	23092	9244	11866
сред.	20644	7655	10266

Таким образом, программа с ассемблерными оптимизациями ускорилась в два раза.