

Memory leak problem

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2006-03/msg01790.html

- *From:* Fernando Barsoba <fbarsoba@xxxxxxxxxxxx>
 - *Date:* Sun, 12 Mar 2006 17:56:46 GMT
-

Hi,

After following the advice received in this list, I have isolated the memory leak problem I am having. I am also using MEMWATCH and I think it is working properly.

The program does some calculations and stores elements in a list. After that, a sorting algorithm is used over that list.

Two functions are called before the sorting process:

```
List = initVoiceChannels(List, TempEvent);  
List = SetTimerT1(List, TempEvent);
```

If I only execute these functions, no memory leak is produced. However, if I call the sorting function **after** any of these functions, the memory leak occurs.

Another test has been to store elements in list and then sorting them. I used the following loop:

```
-----  
typedef struct EVENT  
{  
double codeEvent;  
char descEvent[80];  
double TimeInit;  
double TimeExpire;  
} EVENT;  
-----  
mwInit();  
DLLIST *List = NULL;  
EVENT *TempEvent;  
TempEvent = malloc(1 * sizeof *TempEvent);  
double i=0;  
srand(1);  
for( i = 0; i < 20000; i++ ) {  
TempEvent->codeEvent = i + 1;  
TempEvent->TimeExpire = rand()/1000000;  
DLAddAfter(&List, 0, TempEvent, sizeof *TempEvent);  
TempEvent->codeEvent;  
}  
List = sortElements(List);
```

Memory leak problem

```
DLDestroy(&List);
mwTerm();
exit(0);
```

And there was NO memory leak. So I believe the problem is not in the sorting algorithm per se.

So, going back to the code that does fail. The log from memwatch is:

```
===== MEMWATCH 2.71 Copyright (C) 1992-1999 Johan Lindh =====
```

```
Started at Sun Mar 12 12:52:19 2006
```

```
Modes: __STDC__ 64-bit mWORD==(unsigned long)
mwROUNDALLOC==8 sizeof(mwData)==32 mwDataSize==32
```

```
Stopped at Sun Mar 12 12:52:19 2006
```

```
unfreed: <7> ../dllist.c(51), 32 bytes at 0x470a30 {04 00 00 00 54 31 00 54 00 FE FE FE FE FE FE FE
....T1.T.....}
unfreed: <6> ../dllist.c(45), 20 bytes at 0x4709e8 {00 00 00 00 A8 08 47 00 48 09 47 00 30 0A 47 00
.....G.H.G.0.G.}
unfreed: <5> ../dllist.c(51), 32 bytes at 0x470990 {01 00 00 00 49 4E 49 54 00 FE FE FE FE FE FE FE
....INIT.....}
unfreed: <4> ../dllist.c(45), 20 bytes at 0x470948 {00 00 00 00 E8 09 47 00 00 00 00 00 90 09 47 00
.....G.....G.}
unfreed: <3> ../dllist.c(51), 32 bytes at 0x4708f0 {01 00 00 00 49 4E 49 54 00 FE FE FE FE FE FE FE
....INIT.....}
unfreed: <2> ../dllist.c(45), 20 bytes at 0x4708a8 {00 00 00 00 00 00 00 00 E8 09 47 00 F0 08 47 00
.....G...G.}
```

```
Memory usage statistics (global):
N)umber of allocations made: 13
L)argest memory usage : 344
T)otal of all alloc() calls: 344
U)nfreed bytes totals : 156
MEMWATCH detected 6 anomalies
```

And the code is the following. The list is from the book "C unleashed" from Richard Heathfield et al. chapter 11.

Thanks a lot,

Fernando

```
#include "main.h"
```

```
// FUNCTIONS
```

Memory leak problem

```
// Generate initial event for 'Timer'
DLLIST * SetTimerT1(DLLIST *List, EVENT *TempEvent) {

TempEvent->codeEvent = 4;
strcpy(TempEvent->descEvent, "T1");
TempEvent->TimeInit = MC + Timer;
TempEvent->TimeExpire = 0;

DLAddAfter(&List, 0, TempEvent, sizeof *TempEvent);

MC = MC + Timer;

return List;
}

// Generate Random time
float generateRandomNumber(int intervalA, int intervalB, int incValues) {
float M = intervalA, N = intervalB;
float number = 0;

// incValues is: 0, for neither intervalA nor intervalB included
if( incValues == 0 ) {
do {

//number = M + rand() / (RAND_MAX / (N - M + 1) + 1);
number = M + (N - M) * rand() / ((float) RAND_MAX);
}
while( number == intervalA && number == intervalB );
}
return number;
}

int generateRandomPeriod(float randomNumber, int factor, int typeOfPeriod) {
int X = 0;
int talkspurt = typeOfPeriod;
switch( factor ) {
case 0:
break;
case 1:
X = -(talkspurt)*log(randomNumber);
break;
}
return X;
}

DLLIST * sortElements(DLLIST *List) {
EVENT *First, *Second;
double basketElem1 = 0, basketElem2 = 0;
```

Memory leak problem

```
DLLIST *floor = NULL, *floorP = NULL;
EVENT *floorElem;
int counter1 = 0, counter2 = 0;
List = DLGetFirst(List);
First = DLGetData(List, NULL, NULL);
while(List != NULL) {
    basketElem1 = First->TimeInit;
    if( floor != NULL ) {
        floorP = floor;
        do {
            floorElem = DLGetData(floorP, NULL, NULL);
            basketElem2 = floorElem->TimeInit;
            if( basketElem1 < basketElem2 || basketElem1 == basketElem2) {
                DLAddBefore(&floorP, 0, First, sizeof *First);
                break;
            }
            if( DLGetNext(floorP) == NULL && basketElem1 > basketElem2 ) {
                DLAddAfter(&floorP, 0, First, sizeof *First);
                break;
            }
            floorP = DLGetNext(floorP);
            counter2++;
        } while( floorP != NULL);

        floor = DLGetFirst(floor);
    } else
        DLPrepend(&floor, 0, First, sizeof *First);

    List = DLGetNext(List);
    Second = DLGetData(List, NULL, NULL);
    First = Second;
    counter1++;
}
List = floor;

return List;

}

// Generate first talkspurt for set of voice Channels
DLLIST * initVoiceChannels(DLLIST *Item, EVENT *TempEvent) {
/*
 * Each voice call initiates its first talkspurt at a time Tb randomly
 * chosen between (0,1000) msec
 */
int i = 0;
// temp random numbers
float r1, r2;

int Tb; // time first talkspurt
```

Memory leak problem

```
// for every Channel, generate initial talkspurt
for( i = 0; i < Channels; i++) {
// generate random number [0, 1];
r1 = generateRandomNumber(0, 1, 0);
// printf("generated: %f\n", r1);

// generate random start time (0, 1000)
r2 = generateRandomNumber(0, 1000, 0);
// printf("generated: %f\n", r2);

Tb = r1*r2;

TempEvent->codeEvent=NEWTALKSPURT;
strcpy(TempEvent->descEvent, "INIT");
TempEvent->TimeExpire = 0;

TempEvent->TimeInit = Tb;

DLAddAfter(&Item, 0, TempEvent, sizeof *TempEvent);
}

return Item;
}

DLLIST * storeEvent(int typeEvent, int timeEvent, int timeEventExp, DLLIST *List, EVENT *TempEvent)
{

// Event: Set Talkspurt expiration time
if( typeEvent == 2 ) {
TempEvent->codeEvent = typeEvent;
strcpy(TempEvent->descEvent, "TALKSPURT");
TempEvent->TimeExpire = 0;
TempEvent->TimeInit = timeEvent;

DLLIST *ListP = List;
ListP = DLGetLast(List);

DLAddAfter(&ListP, 0, TempEvent, sizeof *TempEvent);
}

// Event: 5 msec period expires
if( typeEvent == 3 ) {
TempEvent->codeEvent = typeEvent;
strcpy(TempEvent->descEvent, "5MSEC");
TempEvent->TimeInit = timeEvent;
TempEvent->TimeExpire = timeEventExp;

DLAddAfter(&List, 0, TempEvent, sizeof *TempEvent);
}
}
```

Memory leak problem

```
// Event: talkspurt expires
if( typeEvent == 5 ) {
TempEvent->codeEvent = typeEvent;
strcpy(TempEvent->descEvent, "SILENCE");
TempEvent->TimeInit = timeEvent;
TempEvent->TimeExpire = timeEventExp;

DLAddAfter(&List, 0, TempEvent, sizeof *TempEvent);
}

return List;
}

// ***
int main(int argc, char *argv[]) {

mwInit();

if( argc < 3)
printf("Params are not enough");

// check param validity ***

// assign parameters

int param1 = atoi(argv[1]);
int param2 = atoi(argv[2]);

Channels = param1;
Timer = param2;
// ***

// event handling
DLLIST *List = NULL;
EVENT *TempEvent = NULL;

TempEvent = malloc(1 * sizeof *TempEvent);

// initialize voice channels
srand(1);
List = initVoiceChannels(List, TempEvent);

// generate T1 event (variable is 'Timer')
// MC = MC + Timer (2 ms by default)
List = SetTimerT1(List, TempEvent);
printf("Elements in list: %d\n", DLCount(List));
List = sortElements(List);
```

Memory leak problem

Memory leak problem

```
printf("Elements in list: %d\n", DLCount(List));  
free(TempEvent);
```

```
/* Destroy the list */  
DLDestroy(&List);  
mwTerm();  
exit(0);  
}
```

```
#ifndef MAIN_H_  
#define MAIN_H_
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
#include <ctype.h>  
#include <math.h>  
#include <assert.h>
```

```
#include "dlist.h"
```

```
#include "memwatch.h"
```

```
#define NEWTALKSPURT 1;  
#define TALKSPURT 2;  
#define MEANTALKSPURT 400;  
#define MEANSILENCE 600;  
#define MAXMC 100;
```

```
typedef struct EVENT  
{  
int codeEvent;  
char descEvent[10];  
double TimeInit;  
double TimeExpire;  
} EVENT;
```

```
// GLOBAL VARIABLES
```

```
double MC = 0;
```

```
int Timer;  
int Channels;
```

Memory leak problem

```
// ***

DLLIST * SetTimerT1(DLLIST *, EVENT *);
float generateRandomNumber(int, int, int);
int generateRandomPeriod(float, int, int);
DLLIST * sortElements(DLLIST *);
DLLIST * initVoiceChannels(DLLIST *, EVENT *);

#endif /*MAIN_H_*/

/* dllist.c – Double linked list library source
 *
 * DLLIST – Double-Linked List Library
 *
 * Copyright (C) 2000 Richard Heathfield
 * Eton Computer Systems Ltd
 * Macmillan Computer Publishing
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the terms of the GNU General
 * Public License as published by the Free Software
 * Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY WARRANTY; without even the
 * implied warranty of MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public License
 * for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this program; if not, write
 * to the Free Software Foundation, Inc., 675 Mass Ave,
 * Cambridge, MA 02139, USA.
 *
 * Richard Heathfield may be contacted by email at:
 * binary@xxxxxxxxxxxxxxxxxxxxxx
 *
 */

#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include "dllist.h"
```

Memory leak problem

```
DLLIST *DLCreate(int Tag, void *Object, size_t Size)
{
    DLLIST *NewItem;

    NewItem = malloc(sizeof *NewItem);
    if(NewItem != NULL)
    {
        NewItem->Prev = NewItem->Next = NULL;
        NewItem->Tag = Tag;
        NewItem->Size = Size;
        NewItem->Object = malloc(Size);
        if(NULL != NewItem->Object)
        {
            memcpy(NewItem->Object, Object, Size);
        }
        else
        {
            free(NewItem);
            NewItem = NULL;
        }
    }

    return NewItem;
}

int DLInsertBefore(DLLIST *ExistingItem, DLLIST *NewItem)
{
    int Result = DL_SUCCESS;

    if(ExistingItem != NULL && NewItem != NULL)
    {
        NewItem->Next = ExistingItem;
        NewItem->Prev = ExistingItem->Prev;
        ExistingItem->Prev = NewItem;
        if(NewItem->Prev != NULL)
        {
            NewItem->Prev->Next = NewItem;
        }
    }
    else
    {
        Result = DL_NULL_POINTER;
    }

    return Result;
}

int DLInsertAfter(DLLIST *ExistingItem, DLLIST *NewItem)
{
```

Memory leak problem

```
int Result = DL_SUCCESS;

if(ExistingItem != NULL &&NewItem != NULL)
{
NewItem->Prev = ExistingItem;
NewItem->Next = ExistingItem->Next;
ExistingItem->Next = NewItem;
if(NewItem->Next != NULL)
{
NewItem->Next->Prev = NewItem;
}
}
else
{
Result = DL_NULL_POINTER;
}

return Result;
}

int DLPrepend(DLLIST **Item,
int Tag,
void *Object,
size_t Size)
{
int Result = DL_SUCCESS;

DLLIST *p;
DLLIST *Start;

assert(Item != NULL);

p = DLCreate(Tag, Object, Size);

if(p != NULL)
{
if(NULL == *Item)
{
*Item = p;
}
else
{
Start = DLGetFirst(*Item);
DLInsertBefore(Start, p);
}
}
else
{
Result = DL_NO_MEM;
}
}
```

Memory leak problem

```
return Result;
}

int DLAppend(DLLIST **Item,
int Tag,
void *Object,
size_t Size)
{
int Result = DL_SUCCESS;

DLLIST *p;
DLLIST *End;

assert(Item != NULL);

p = DLCreate(Tag, Object, Size);

if(p != NULL)
{
if(NULL == *Item)
{
*Item = p;
}
else
{
End = DLGetLast(*Item);
DLInsertAfter(End, p);
}
}
else
{
Result = DL_NO_MEM;
}

return Result;
}
```

```
/* Add new item immediately after current item */
int DLAddAfter(DLLIST **Item,
int Tag,
void *Object,
size_t Size)
{
int Result = DL_SUCCESS;
DLLIST *p;

assert(Item != NULL);

p = DLCreate(Tag, Object, Size);
```

Memory leak problem

```
if(p != NULL)
{
if(NULL == *Item)
{
*Item = p;
}
else
{
DLInsertAfter(*Item, p);
}
}
else
{
Result = DL_NO_MEM;
}

return Result;
}
```

/* Add new item immediately before current item */

```
int DLAddBefore(DLLIST **Item,
int Tag,
void *Object,
size_t Size)
{
int Result = DL_SUCCESS;
DLLIST *p;

assert(Item != NULL);

p = DLCreate(Tag, Object, Size);

if(p != NULL)
{
if(NULL == *Item)
{
*Item = p;
}
else
{
DLInsertBefore(*Item, p);
}
}
else
{
Result = DL_NO_MEM;
}

return Result;
}
```

Memory leak problem

```
int DLUpdate(DLLIST *Item,
int NewTag,
void *NewObject,
size_t NewSize)
{
int Result = DL_SUCCESS;

void *p;

if(NewSize > 0)
{
p = realloc(Item->Object, NewSize);
if(NULL != p)
{
Item->Object = p;
memmove(Item->Object, NewObject, NewSize);
Item->Tag = NewTag;
Item->Size = NewSize;
}
else
{
Result = DL_NO_MEM;
}
}
else
{
Result = DL_ZERO_SIZE;
}

return Result;
}

void *DLGetData(DLLIST *Item,
int *Tag,
size_t *Size)
{
void *p = NULL;

if(Item != NULL)
{
if(Tag != NULL)
{
*Tag = Item->Tag;
}
if(Size != NULL)
{
*Size = Item->Size;
}
p = Item->Object;
}
}
```

Memory leak problem

```
return p;
}

DLLIST *DLExtract(DLLIST *Item)
{
if(Item != NULL)
{
if(Item->Prev != NULL)
{
Item->Prev->Next = Item->Next;
}
if(Item->Next != NULL)
{
Item->Next->Prev = Item->Prev;
}
Item->Prev = Item->Next = NULL;
}
return Item;
}

void DLDelete(DLLIST *Item)
{
if(Item != NULL)
{
DLExtract(Item);

if(Item->Object != NULL)
{
free(Item->Object);
}
free(Item);
}
}

/* Exchange two items. Both must be non-NULL. */
int DLExchange(DLLIST *ItemA, DLLIST *ItemB)
{
int Result = DL_SUCCESS;
DLLIST *t0;
DLLIST *t1;
DLLIST *t2;
DLLIST *t3;

if(ItemA != NULL && ItemB != NULL)
{
if(ItemA->Next == ItemB)
{
DLExtract(ItemA);
DLInsertAfter(ItemB, ItemA);
}
else if(ItemB->Next == ItemA)
```

Memory leak problem

```
{
DLExtract(ItemB);
DLInsertAfter(ItemA, ItemB);
}
else
{
t0 = ItemA->Prev;
t1 = ItemA->Next;
t2 = ItemB->Prev;
t3 = ItemB->Next;

DLExtract(ItemA);
DLExtract(ItemB);

if(t2 != NULL)
{
DLInsertAfter(t2, ItemA);
}
else
{
DLInsertBefore(t3, ItemA);
}

if(t0 != NULL)
{
DLInsertAfter(t0, ItemB);
}
else
{
DLInsertBefore(t1, ItemB);
}
}
else
{
Result = DL_NULL_POINTER;
}

return Result;
}

void DLDestroy(DLLIST **List)
{
DLLIST *Marker;
DLLIST *Prev;
DLLIST *Next;

if(*List != NULL)
{
/* First, destroy all previous items */
Prev = (*List)->Prev;
```

Memory leak problem

```
while(Prev != NULL)
{
Marker = Prev->Prev;
DLDelete(Prev);
Prev = Marker;
}
```

```
Next = *List;
do
{
Marker = Next->Next;
DLDelete(Next);
Next = Marker;
} while(Next != NULL);
*List = NULL;
}
```

```
DLLIST *DLGetPrev(DLLIST *List)
{
if(List != NULL)
{
List = List->Prev;
}

return List;
}
```

```
DLLIST *DLGetNext(DLLIST *List)
{
if(List != NULL)
{
List = List->Next;
}

return List;
}
```

```
DLLIST *DLGetFirst(DLLIST *List)
{
if(List != NULL)
{
while(List->Prev != NULL)
{
List = List->Prev;
}
}
return List;
}
```

```
DLLIST *DLGetLast(DLLIST *List)
```

Memory leak problem

```
{
if(List != NULL)
{
while(List->Next != NULL)
{
List = List->Next;
}
}
return List;
}
```

```
DLLIST *DLJoin(DLLIST *Left, DLLIST *Right)
{
if(Left != NULL && Right != NULL)
{
Left = DLGetLast(Left);
Right = DLGetFirst(Right);

Left->Next = Right;
Right->Prev = Left;
}

return DLGetFirst(Left);
}
```

```
int DLCount(DLLIST *List)
{
int Items = 0;

DLLIST *Prev = List;
DLLIST *Next = List;

if(List != NULL)
{
++Items;
while((Prev = DLGetPrev(Prev)) != NULL)
{
++Items;
}
while((Next = DLGetNext(Next)) != NULL)
{
++Items;
}
}

return Items;
}
```

```
int DLWalk(DLLIST *List,
int(*Func)(int, void *, void *),
```

Memory leak problem

```
void *Args)
{
DLLIST *ThisItem = List;
int Result = 0;

if(List != NULL)
{
for(ThisItem = DLGetFirst(List);
0 == Result && ThisItem != NULL;
ThisItem = ThisItem->Next)
{
Result = (*Func)(ThisItem->Tag,
ThisItem->Object,
Args);
}
}
return Result;
}

/* end of dllist.c */
```