18ES611 Embedded System Programming

Sarath tv

Memory Leak and Fragmentation

Memory leak: With or without garbage collection, it is possible for a program to inadvertently **accumulate memory** that is **never freed**.

```
/* Function with memory leak */
#include <stdlib.h>

void f()
{
    int *ptr = (int *) malloc(sizeof(int));

    /* Do some work */

    return; /* Return without freeing ptr*/
}
```

To avoid memory leaks, memory allocated on heap should always be freed when no longer needed.

```
/* Function without memory leak */
#include <stdlib.h>;

void f()
{
    int *ptr = (int *) malloc(sizeof(int));

    /* Do some work */

    free(ptr);
    return;
}
```

Memory Leak

```
void func(){
    char *ch;
    ch = (char*) malloc(10);
}
```

//ch not valid outside, no way to access malloc-ed memory

Char-ptr ch is a local variable that goes out of scope at the end of the function, leaking the dynamically allocated 10 bytes. There is no way to access (or free it) now, as there are no ways to get to it anymore.

General Advice for Memory Leaks

Make sure your dynamically allocated memory does in fact get freed.

Don't allocate memory and forget to assign the pointer.

Don't overwrite a pointer with a new one unless the old memory is freed.

Memory fragmentation

Memory fragmentation: This occurs when a program **chaotically allocates** and **deallocates memory** in **varying sizes**.

A fragmented memory has allocated and **free memory chunks interspersed**, and often the **free memory chunks become too small to use**.

Defragmentation is required. Defragmentation and garbage collection are both very problematic for real-time systems.

For this example, it is assumed there is a 10K heap. First, an area of 3K is requested, thus:

```
#define K (1024)

char *p1;

p1 = malloc(3*K);
```

Then, a further 4K is requested

```
p2 = malloc(4*K);
```

3K of memory is now free.

Some time later, the first memory allocation, pointed to by p1, is de-allocated:

```
free(p1);
```

This leaves 6K of memory free in two 3K chunks. A further request for a 4K allocation is issued:

```
p1 = malloc(4*K);
```

This results in a failure – NULL is returned into p1 – because, even though 6K of memory is available, there is not a 4K contiguous block available. This is memory fragmentation.

Suppose for a simple toy example that you have ten bytes of memory:

```
|   |   |   |   |   |   |   |   |   |   |
  0   1   2   3   4   5   6   7   8   9
```

Now let's allocate three three-byte blocks, name A, B, and C:

```
| A | A | A | B | B | B | C | C | C |   |
  0   1   2   3   4   5   6   7   8   9
```

Now deallocate block B:

```
| A | A | A |   |   |   | C | C | C |   |
  0   1   2   3   4   5   6   7   8   9
```

Tools

Cppcheck

Valgrind

splint

RATS

SMATCH

THANK YOU!!!!!