Lab 6
Dynamic Memory Management

CSCI 4061
Introduction to Operating Systems

Oct 16, 2023

Overview

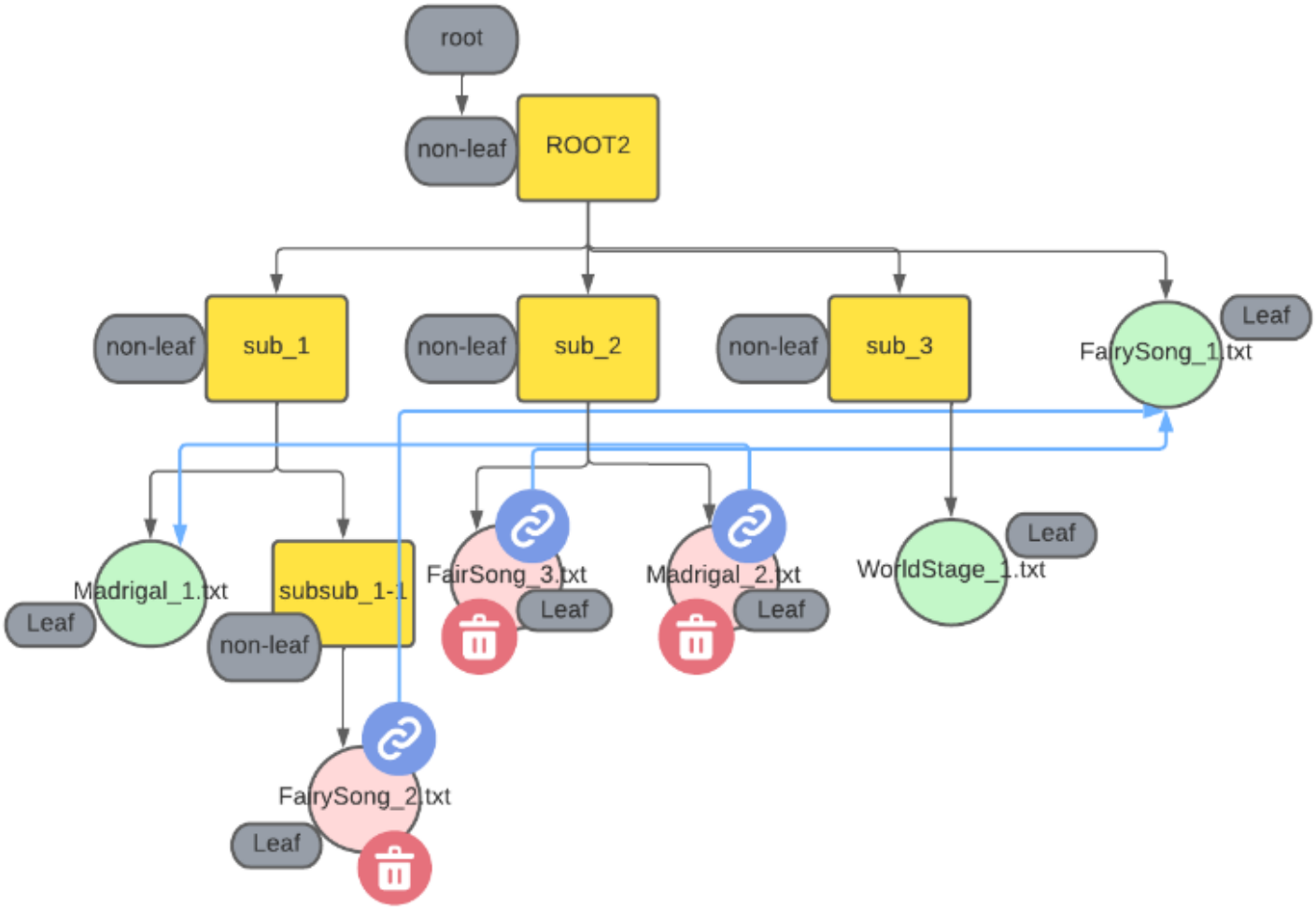
- PA2 Discussion
- Programming Exercise
- Memory allocation
- Valgrind

PA2 - Data deduplication

Project Goal

1. Reduce the copies of data across machines and distributed file system to increase storage capacity
2. Given a root folder with nested folders and files
3. Copies of multiple files exists in the given folder hierarchy
4. Find all copies and only maintain one
5. The copies should be converted to symbolic link pointing to the maintained file

PA2 - Data deduplication



PA2 - Data deduplication

Template

- *root_process.c*
  1. Creates the first non-leaf process (for traversing root directory)
  2. The first non leaf process communicates all the aggregated file-hash from its children to the root process via a pipe
  3. The root process maintains unique files (one with smaller number) and deletes all duplicate files
  4. The paths of duplicate files points to the maintained file corresponding to them
  5. The root process will redirect stdout to given output file and report the symbolic link paths and its content (path to the original file)

PA2 - Data deduplication

Template

- *nonleaf_process.c*
  1. Traverse the given directory and performs two tasks
     - If the entry is a directory, create a child process and call exec on nonleaf_process with the directory entry
     - If the entry is a file, create a child process and call exec on leaf_process with the file entry
  2. The children should communicate the aggregated file-hash with the nonleaf_process via pipe
  3. The non-leaf process should aggregate all the file-hash from children and send it to its parent process via pipe

PA2 - Data deduplication

Template

- *leaf_process.c*
  1. leaf process is only called on files
  2. The leaf process will create hash from the given file using hash_data_block (hash.h)
  3. Send the file-hash to the parent process via pipe

PA2 - Data deduplication

Timeline

1. Intermediate submission: Oct 20, 11:59 pm
2. Final submission: Oct 27, 11:59 pm

Programming Exercise

Activity

1. You are given a program with multiple memory bugs
2. Resolve the bugs
3. Run valgrind on your code and copy results to a new file output.txt

We will cover memory allocation, memory leaks and using valgrind to identify memory leaks.

Memory Allocation

Static and dynamic allocation

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]){
    int arr_stat[10]; // static allocation of an array of size 10
    int *arr_dyn = (int *) malloc (sizeof(int) * 10); // dynamic allocation
                                                      // of an array of size 10
    for(int i = 0; i < 10; i++){
        arr_stat[i] = i;
        arr_dyn[i] = i;
    }

    printf("static size - %d\n", sizeof(arr_stat));
    printf("dynamic size - %d\n", sizeof(arr_dyn));

    free(arr_dyn); // frees allocated heap memory
    arr_dyn = NULL; // avoids double free
    return 0;
}
```

Memory Allocation

Terminal (p1.c)

```
$ make lab6_p1
./main
static size - 40
dynamic size - 8 # Reason for many errors in PA1
```

Memory Allocation

Terminal (p1.c)

```
$ make lab6_p1
./main
static size - 40
dynamic size - 8 # Reason for many errors in PA1
```

Solution

Malloc can be called only if we know size during runtime.
Keep track of this.
For strings you can use strlen() to get size of a char array

Memory allocation

Stack smashing

1. Stack grown downwards and Heap grows upwards in program memory layout
2. Storing more data than the memory allocated to a stack variable can cause buffer overflow access
3. You may end up corrupting adjacent data on the stack

Stack smash

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]){
    int arr[10];
    for(int i = 0; i < 11; i++){ // buffer overflow access
        arr[i] = i;
    }

    return 0;
}
```

Memory Allocation

Terminal (p2.c)

```
$ make lab6_p2
./main
*** stack smashing detected ***: terminated
make: *** [Makefile:7: lab6_p2] Aborted (core dumped)
```

Memory Allocation

Terminal (p2.c)

```
$ make lab6_p2
./main
*** stack smashing detected ***: terminated
make: *** [Makefile:7: lab6_p2] Aborted (core dumped)
```

Solution

Always check the memory size before storing the data to a buffer (bound check)

Memory allocation

Memory leaks / heap overflow

1. Heap memory allocations (malloc) should be explicitly freed in C
2. Unlike stack overflow, heap overflows are not reported as errors normally

Memory leak

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]){
    int *arr = (int *) malloc (sizeof(int) * 10);
    for(int i = 0; i < 11; i++){ // silent memory overflow
        arr[i] = i;
    }

    arr = NULL; // memory leak
    return 0;
}
```

Memory allocation

1. Memory leak identification

Use a memory mismanagement detector like Valgrind

Usage:
Compile you code with $-g$ flag

```
valgrind --leak-check=full ./exe
```

Terminal (p3.c)

```
$ make lab6_p3_2
==363579== Invalid write of size 4
==363579==    at 0x1091AA: main (p3.c:7)
==363579==  Address 0x4a99068 is 0 bytes after a block of size 40 alloc'd
==363579==    at 0x4848899: malloc (in /usr/libexec/valgrind/
                                    vgpreload_memcheck-amd64-linux.so)
==363579==    by 0x109185: main (p3.c:5)
==363579==
==363579==
==363579== HEAP SUMMARY:
==363579==     in use at exit: 0 bytes in 0 blocks
==363579==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
```

Memory allocation

2. Heap overflow error reporting

Use $-g$ flag and $-fsanitize = address$ when you compile code

Terminal (p3.c)

```
$ make lab6_p3_3
./main
=================================================================
==371074==ERROR: AddressSanitizer: heap-buffer-overflow on address
                 0x604000000038 at pc 0x559c2785e23f bp
                 0x7ffd214eccf0 sp 0x7ffd214ecce0
WRITE of size 4 at 0x604000000038 thread T0
#0 0x559c2785e23e in main /home/csci4061/Lab6/code/sample/p3.c:7
#1 0x7fd58b01cd8f in __libc_start_call_main
                        ../sysdeps/nptl/libc_start_call_main.h:58
#2 0x7fd58b01ce3f in __libc_start_main_impl ../csu/libc-start.c:392
#3 0x559c2785e104 in _start
                        (/home/csci4061/Lab6/code/sample/main+0x1104)
```

Programming Exercise - Expected output

Terminal

```
$ make
==392853== Memcheck, a memory error detector
==392853== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==392853== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==392853== Command: ./main 3 3
==392853==
0 0 0
0 0 0
0 0 0
==392853==
==392853== HEAP SUMMARY:
==392853==     in use at exit: 0 bytes in 0 blocks
==392853==   total heap usage: 9 allocs, 9 frees, 1,144 bytes allocated
==392853==
==392853== All heap blocks were freed -- no leaks are possible
==392853==
==392853== For lists of detected and suppressed errors, rerun with: -s
==392853== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Deliverables

Individual submission: Zip and submit to Gradescope by Oct 17, 11:59pm

1. bugs.c
2. output.txt