CSE 451: Operating Systems

Section 3

Project 0 recap, Project 1

Andrew Tanenbaum talk

Andrew Tanenbaum talk

- Microkernels
  - Tanenbaum-Torvalds debate: http://oreilly.com/catalog/opensources/book/appa.html
- Software bloat
  - Is software really getting slower faster than hardware is getting faster?

Project 0: queue problems

- Must check for empty queues before reversing or sorting
- Should test on several queues
  - Short, long
  - Randomized order

Project 0: common problem #1

- Linear probing misunderstandings
  - Must mark cells as *vacated* (different than free)
- Consider hash table size of 10
  - Insert key1 -> hash = 5 ; Insert key2 -> hash = 15
  - Occupy positions 5 & 6
  - Delete key1
  - Lookup key2: 5 is empty but need to look at 6 also

Project 0: common problem #2

- Properly handling set_hash_function()
- Consider the following sequence:
  - Insert key1 -> hash = 5 under hash function *a*
  - Set hash function to *b* such that key1 -> hash = 6 under hash function *b*
  - Look up key1, turns out to be empty!

Project 0: common problem #2

- Solutions?
  - Rehash
  - Prevent user from changing hash function if hash table is non-empty

Project 0: other problems

- Resizing hash table
- Using int or char as key type instead of general type (void *)
- Memory leaks

Coding style

- Describe the *interface* when declaring functions in .h files
  - What does it do?
  - What assumptions does it make about its arguments?
  - What does it return?
  - How does it indicate an error condition?

Coding style

- Write comments for tricky implementation sections:
  - Bad comment:

```
somePtr = NULL;  // Set somePtr to NULL
```

  - Good comment:

```
somePtr = NULL;  // Always reset the
                 // pointer to NULL
                 // after the shared
                 // memory it points to
                 // has been freed
```


Coding style

- Always use header guards:

```
#ifndef _HASH_TABLE_H
#define _HASH_TABLE_H

// header file code here...

#endif  /* _HASH_TABLE_H */
```


Coding style

- Properly indent nested blocks
  - man 1 indent
  - Let your text editor do it for you!

Coding style

- Be consistent with your naming
  - Functions: pick a style and stick to it
    - set_hash_function() style is ok
    - SetHashFunction() style also ok
  - End typenames in _t

```
typedef foo_struct * foo_t;
```

  - Choose reasonable variable names

```
int n_comp_conns;                // BAD
int num_completed_connections;   // GOOD
```


Memory management

```
void do_stuff(char *buf, int len) {
   ...
   free(buf);
}

int main() {
   char *mybuf =
       (char *)malloc(LEN*sizeof(char));
   do_stuff(mybuf, LEN);
   ...
   free(mybuf);     // Double free: undefined
                    // behavior!
}
```


Memory management

- Always be explicit about who owns memory
  - If a function allocates some memory that the caller must free, say so!
  - If a function frees some memory that the caller should no longer use, say so!
- Define pairs of allocate and free functions
  - Ideally, whoever calls allocate function also calls free function; if not, carefully consider usage

Advanced memory mgmt.

- What if multiple processes or threads are accessing the same structure in memory?
  - When can we free?
    - Reference counting
- How does memory management within the kernel differ?
  - Slab allocator [Bonwick '94]

Project 1

Project 1

- Due Monday at 11:59pm!
- Include all group members & group letter in write-up
- Follow same turnin instructions again
  - Only one team member needs to run turnin

Project 1 turnin

- Preserve directories when submitting changed files
  - When we extract your changed files, they should go to the right directory, so it is unambiguous which file you changed
  - This is easy to do with **tar** command
- Writeup requires a list of modified files (#3): please use full path name

Project 1 notes

- Special functions should be used to copy data between user space and kernel
  - Why?
  - access_ok(), copy_from_user(), copy_to_user(): look for example usage in kernel
    - Definition, gory details: arch/i386/lib/usercopy.c

Project 1 notes

- Where does printk() output go?
  - Possibly to console
    - include/linux/kernel.h: defines KERN_XYZ log levels
  - **dmesg** command
  - /var/log/messages

Project 1 tips

- Re-read the project description for hints
- Read the man pages!
- Navigating Linux kernel code: see Section 2
- Get started!!