CSE 451 Section 3:

Project0 highlights, File descriptors

Overview

- Project 0 highlights
- A bit more on project 1 (kernel devel part)
- File descriptors

Project 0 highlights

- Most frequent issue:
  Clean up before exiting main
- Memory leak in queue_remove

- Revise hash function properties

Cleanup

- Why should we clean up?
- What should we clean up?

Cleanup

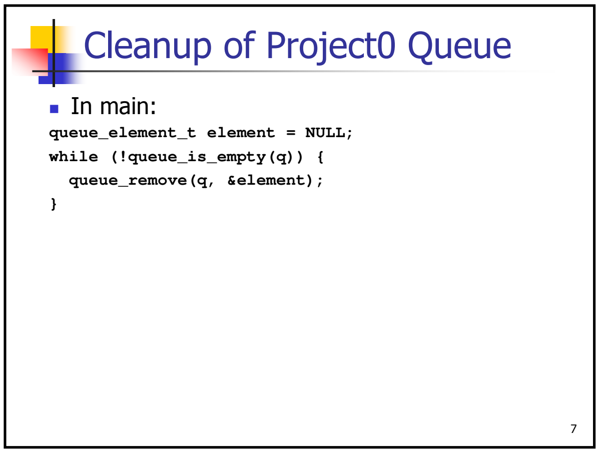
We should clean up because:

- Always enforce rigorous programming style
- Sometimes the underlying system doesn't release resources right away
  - Processes in process pools
  - Ports in older versions of Linux
  - Fork doesn't clean up
- Debugging
  - E.g., debugging other memory leaks
- Sometimes others need cleanup info
  - E.g., other Bittorrent nodes, chunk servers in GFS

Cleanup

We should clean up anything that we allocate:

- Dynamically allocated memory
- Open file descriptors
- Open ports
- Open network connections
- Release locks on files, delete lock files

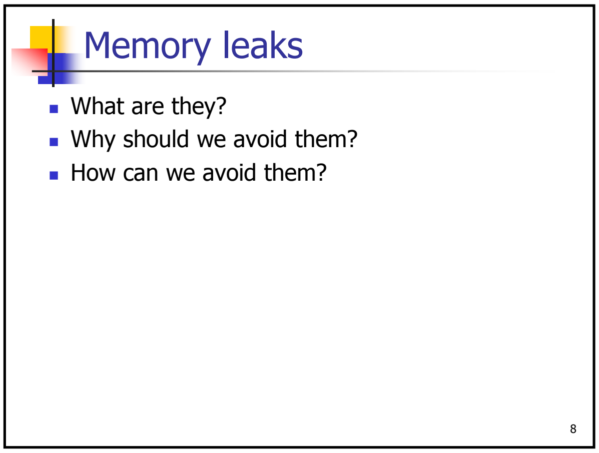


Cleanup of Project0 Queue

- In main:

```
queue_element_t element = NULL;
while (!queue_is_empty(q)) {
  queue_remove(q, &element);
}
```

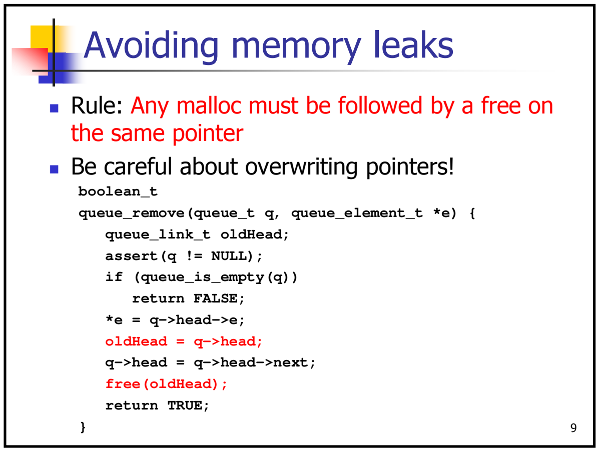
7



Memory leaks

- What are they?
- Why should we avoid them?
- How can we avoid them?

8

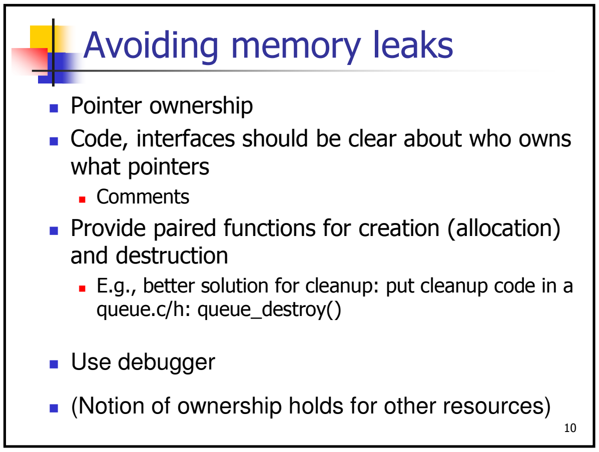


Avoiding memory leaks

- Rule: Any malloc must be followed by a free on the same pointer
- Be careful about overwriting pointers!

```
boolean_t
queue_remove(queue_t q, queue_element_t *e) {
   queue_link_t oldHead;
   assert(q != NULL);
   if (queue_is_empty(q))
      return FALSE;
   *e = q->head->e;
   oldHead = q->head;
   q->head = q->head->next;
   free(oldHead);
   return TRUE;
}
```

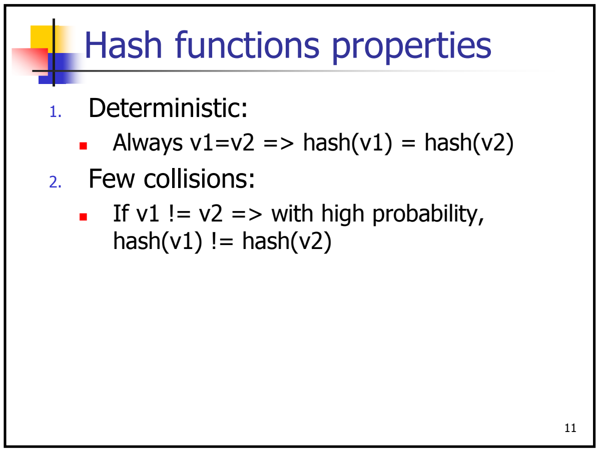
9



Avoiding memory leaks

- Pointer ownership
- Code, interfaces should be clear about who owns what pointers
  - Comments
- Provide paired functions for creation (allocation) and destruction
  - E.g., better solution for cleanup: put cleanup code in a queue.c/h: queue_destroy()
- Use debugger
- (Notion of ownership holds for other resources)

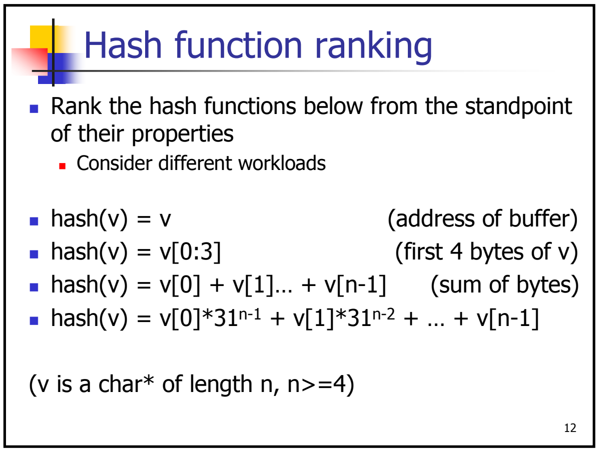
10



Hash functions properties

1. Deterministic:
   - Always v1=v2 => hash(v1) = hash(v2)
2. Few collisions:
   - If v1 != v2 => with high probability, hash(v1) != hash(v2)

11



Hash function ranking

- Rank the hash functions below from the standpoint of their properties
  - Consider different workloads

- hash(v) = v (address of buffer)
- hash(v) = v[0:3] (first 4 bytes of v)
- hash(v) = v[0] + v[1]… + v[n-1] (sum of bytes)
- hash(v) = v[0]*31^{n-1} + v[1]*31^{n-2} + … + v[n-1]

(v is a char* of length n, n>=4)

12

Overview

- Project 0 highlights
- A bit more on project 1 (kernel devel part)
- File descriptors

Kernel development steps:

- Modify the kernel
- Build the kernel image on forkbomb

```
make bzImage
```

- Transfer the bzImage to the Linux guest
  - scp it to /boot
- Boot your new Linux kernel in VMware
  - choose bzImage

Execcounts tips

- Look at examples of system calls
  - E.g., getpid, kill, write
- Find and read online tutorials and examples
- Be very careful at translating addresses from userspace to kernel space

Overview

- Project 0 highlights
- A bit more on project 1 (kernel devel part)
- File descriptors

Process structure (task_struct)

```
struct task_struct {
   volatile long state;            // running, blocked, stopped, zombie
   unsigned long flags;
   long priority;                  // scheduling priority
   long counter;                   // before re-scheduling
   unsigned long policy;           // sched policy: FIFO, round-robin, etc.
   struct task_struct *next_task, *prev_task;    // doubly-linked list
   struct task_struct *next_run, *prev_run;
   int exit_code;
   int pid;
   struct task_struct *p_pptr;    // pointer to parent process
   unsigned long start_time;
   usingned short uid, gid;
   struct files_struct *files;     //
   struct mm_struct *mm;           // memory management
}
```

(Above is incomplete and approximate)

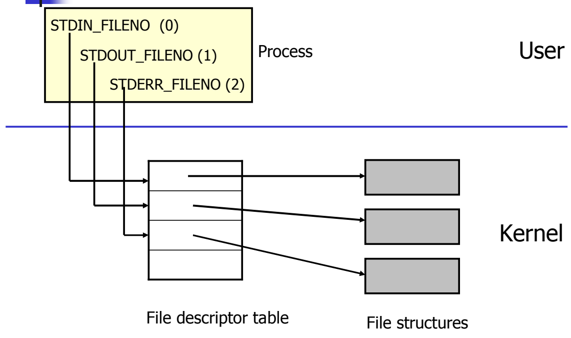
File descriptors

- What is a file descriptor?
- Examples?
- What happens on fork with open file ?

File descriptors

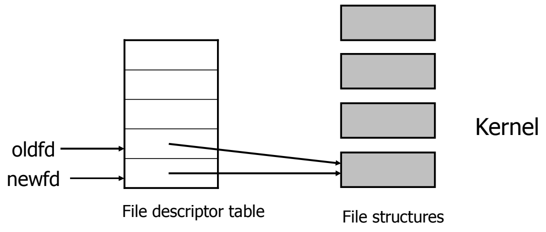
- A file descriptor is an index in the file table for the current process
  - Each entry contains a pointer to a kernel structure storing the file's info, file cursor position, flags, etc.
- Examples:
  - Files, directories
  - STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO
  - Block/character devices
  - Sockets
  - *Pipes*

File descriptor table

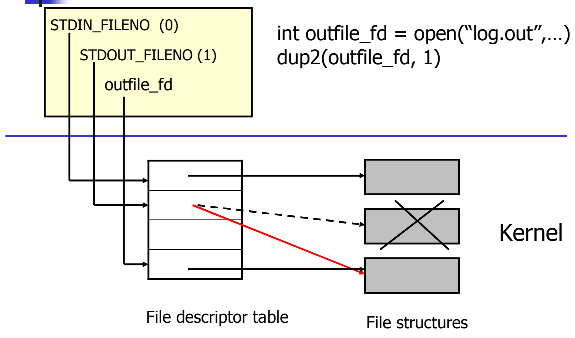


Duplicating file descriptors

- Man dup2: dup2(old_fd, new_fd)
  - Duplicates the old_fd entry in the process' file table into the new_fd entry



Example: redirecting stdout



int outfile_fd = open("log.out",…)
dup2(outfile_fd, 1)