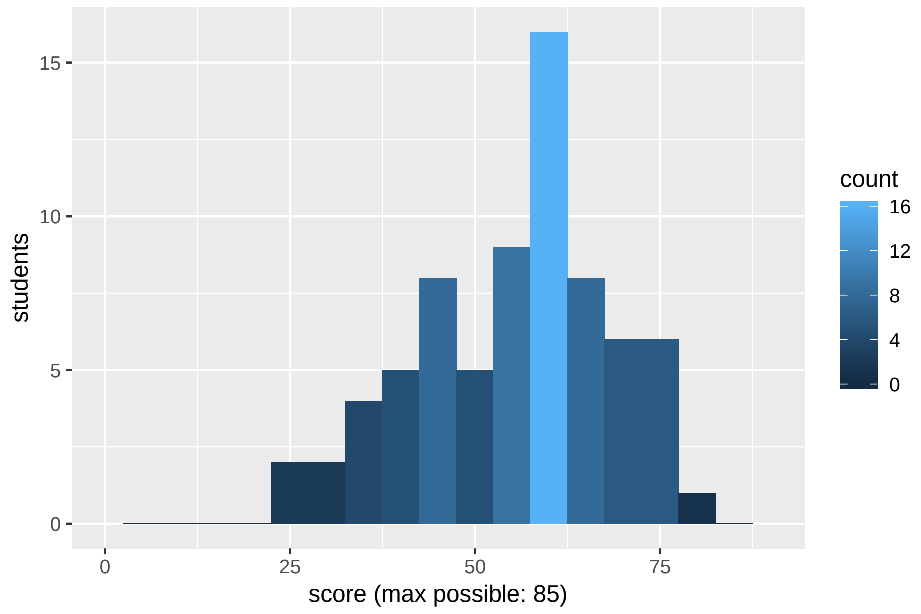
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.828 Fall 2018

Quiz II Solutions

Mean 55.4   Median 58   Standard deviation 13.0



I Lab 4

You just completed the user-level implementation of fork() using `sys_exofork()` for Lab 4. Your friend who is also taking 6.828 is advertising her idea for even faster COW page allocation; *only duplicate the writable pages during COW*. This is her implementation:

```
envid_t
fork(void)
{
    envid_t envid;
    set_pgfault_handler(pgfault);

    // Create a child.
    envid = sys_exofork();
    if (envid < 0)
        return envid;
    if (envid == 0) {
        thisenv = &envs[ENVX(sys_getenvid())];
        return 0;
    }

    // My COW opt starts here!
    int pn, end_pn, r;

    for (pn = 0; pn < PGNUM(UTOP); ) {
        if (!(uvpd[pn >> 10] & PTE_P)) {
            pn += NPTENTRIES;
            continue;
        }
        for (end_pn = pn + NPTENTRIES; pn < end_pn; pn++) {
            // Here: skip non-writeable pages
            if ((uvpt[pn] & (PTE_P|PTE_U|PTE_W)) != (PTE_P|PTE_U|PTE_W))
                continue;
            if (pn == PGNUM(UXSTACKTOP - 1))
                continue;
            duppage(envid, pn);
        }
    }
```

1. [5 points]: Briefly explain what problems you see with her solution and how you would address them.

Answer: Her implementation doesn't map the RO pages at all (or COW pages, for that matter). Those should be mapped too, with the proper permissions, otherwise the child won't be able to execute.

II Lab 5

In Lab 5, the file server services requests in a loop (see `serve()` in `fs/serv.c`):

```
while (1) {
    perm = 0;
    req = ipc_recv((int32_t *) &whom, fsreq, &perm);

    // service request

    sys_page_unmap(0, fsreq);
}
```

Note that the `ipc_recv()` call and `sys_page_unmap()` call reference the same hard-coded fsreq address (`0x0ffff000`).

2. [2 points]: If we omit the call `to sys_page_unmap()`, will the code still behave correctly? That is, will the file server still correctly satisfy requests? Explain briefly your answer. (If not, make sure to explain what will go wrong.)

Answer: Yes, it will work correctly.

3. [5 points]: If we omit the call to `sys_page_unmap()`, will the kernel leak physical memory? Explain briefly why or why not?

Answer: Th kernel won't leak memory. A correct kernel wouldn't let any user process leak physical memory. Here, when ipc_recv() maps a new physical page onto an already-mapped virtual address, page_insert() calls page_remove(), which calls page_decref(), which frees the physical page if necessary.

III Journaling ext2fs

This question is based on the paper *Journaling the Linux ext2fs Filesystem* by Tweedie.

4. [5 points]: Give an example of a file system call that does *not* need journaling and briefly explain why.

Answer: Any file system call that doesn't modify the file system (e.g., `stat`).

5. [5 points]: When an new transaction overwrites data written by a previous transaction that is committing, ext2fs journaling causes a page fault and makes new copy of the page. Describe a concrete scenario what could go wrong (i.e., the on-disk file system ends up in an incorrect state) if extfs journaling didn't do this.

Answer: A directory may contain data of the new transaction, even if the new transaction doesn't commit. In general, if tx1 and tx2 both modify page X. now, when tx data is getting written to disk, tx1's page X is the modified tx2 version. then, tx1 commits (and tx2 does not), system crashes, and replaying the log uses the bad copy of page X (because it has been modified by tx2).

IV OS organization

6. [8 points]: Which of the following statements are true about kernel organizations, based on *Exokernel: an operating system architecture for application-level resource management* by Engler et al.?

(Circle True or False for each choice.)

- **True / False** The JOS file system is implemented in monolithic style
- **True / False** The UVPT trick in JOS is an example use of an exokernel-style interface
- **True / False** A monolithic kernel implements `fork` in the kernel
- **True / False** User programs in an exokernel can modify other user program's kernel state

Answer: False; the JOS file runs as a user-level server. True; the page tables are directly exposed to user space for UVPT. True; for example, see xv6. False; user programs cannot directly modify kernel state without permission.

V Biscuit

7. [8 points]:

Which of the following statements are true about Biscuit, according to *The benefits and costs of writing a POSIX kernel in a high-level language* by Cutler et al.?

(Circle True or False for each choice.)

- **True / False** Interrupt handlers can use any Go feature
- **True / False** Lock-free code can call functions that may cause the caller to be put to sleep
- **True / False** Biscuit needs less kernel memory than Linux to run efficiently
- **True / False** A Biscuit system call returns an error when the system call exhausts the available kernel memory.

Answer: False; to avoid deadlock Biscuit interrupt handlers just set a a flag to wake up a go routine. True; Biscuit can go to sleep in an RCU section. False; Biscuit needs headroom. False; Biscuit will block in a system call until enough memory is available.

VI IX

This question is based on the paper *IX: A Protected Dataplane Operating System for High Throughput and Low Latency* by Tweedie.

8. [5 points]: The Linux kernel is in the control plane instead of in one of the data planes. Briefly explain why it is not in the data plane and what its function is in the control plane.

Answer: Linux kernel is too slow to be in the data plane but is necessary to set up the data planes.

9. [5 points]: Explain why the design of the JOS network stack is unable to achieve the performance that IX delivers. In your explanation identify a concrete design aspect and explain why that aspect limits performance.

Answer: Many system calls per packet, many IPCs per packet, one NIC queue, no parallel processing of connections, etc.

Consider a machine with 12 cores and 10 Gbit/s NIC. Assume an IX configuration with 8 NIC queues and 8 cores, each associated with one queue. Ben writes a server using IX that receives small images (32 Kbyte in size) and that writes each image into a separate file. He tests this configuration with 8 clients, each doing one image transfer per second. You can assume that each client hashes to a different NIC queue. He also runs a computationally-intensive image processing application on Linux that takes a few seconds to process each file in parallel with receiving them; the application uses the remaining 4 cores in Ben's machine.

10. [5 points]: Ben compares the performance of the IX setup with stock Linux (i.e., without IX) on the same hardware. To his surprise, he finds that stock Linux achieves higher CPU utilization than IX and processes images at higher rate than IX. Explain why.

Answer: The IX cores will be busy-spinning waiting for an image to arrive on the network, so they cannot be used for image processing. By contrast, in Linux any cores that are not needed for network handling can be used for image processing. The image processing is computationally-intensive enough for Linux to get higher throughput by leveraging these extra cores.

VII Synchronization

For these questions, refer to McKenny et al. *RCU Usage In The Linux Kernel: One Decade Later*.

For the next question, consider the following simplified banking application, which includes `total()`, a function to calculate the bank's total assets, and `transfer()`, a function to move money from one account to another.

```
rwlock_t lock;

struct account {
  const char *name;
  int balance;
};

struct account *accs[N];

int total(void) {
  int n, total = 0;
  read_lock(&lock);
  for (i = 0; i < N; i++)
    total += accs[i]->balance;
  read_unlock(&lock);
  return total;
}

void transfer(int fromidx, int toidx, int amount) {
  write_lock(&lock);
  accs[fromidx]->balance -= amount;
  accs[toidx]->balance += amount;
  write_unlock(&lock);
}
```

11. [5 points]: Alyssa suggests that RCU can sometimes be a more optimal alternative to read-write locks. In terms of frequency of `total()` and `transfer()` API usage, which load patterns would benefit the most from this potential optimization? Justify your answer.

Answer: RCU would improve performance the most when `total()` is called very often and `transfer()` is called infrequently.

Recall the Linux RCU API you learned about in lecture. A summary is as follows:

- **rcu_read_lock()** Begin an RCU critical section.
- **rcu_read_unlock()** End an RCU critical section.
- **synchronize_rcu()** Wait for pending RCU critical sections to finish.
- **rcu_dereference()** Signal the intent to dereference a pointer inside an RCU critical section.
- **rcu_assign_pointer()** Assign a value to a pointer that is read in RCU critical sections.

So far, Ben has modified the banking application to use RCU as follows:

```
spinlock_t lock;

struct account {
  const char *name;
  int balance;
};

__rcu struct account *accs[N];

int total(void) {
  int n, total = 0;
  rcu_read_lock();
  for (i = 0; i < N; i++)
    total += rcu_dereference(accs[i])->balance;
  rcu_read_unlock();
  return total;
}

void transfer(int fromidx, int toidx, int amount) {
  struct account *newfrom = malloc(sizeof(*newfrom));
  struct account *newto = malloc(sizeof(*newto));
  struct account *oldfrom, *oldto;

  spin_lock(&lock);
  // make a copy of the RCU objects
  oldfrom = rcu_dereference(accs[fromidx]);
  oldto = rcu_dereference(accs[toidx]);
  *newfrom = *oldfrom;
  *newto = *oldto;

  // update the balances
  newfrom->balance -= amount;
  newto->balance += amount;

  // update the RCU objects
  rcu_assign_pointer(accs[fromidx], newfrom);
  rcu_assign_pointer(accs[toidx], newto);
  spin_unlock(&lock);

  // free the previous version of the RCU objects
  synchronize_rcu();
  free(oldfrom);
  free(oldto);
}
```

12. [5 points]: One invariant of the banking application is that the precise output of `total()` should never be changed by `transfer()` because it only moves money between accounts within the same bank. Surprisingly, Ben discovers that `total()` returns several different values during testing it against concurrent invocations of `transfer()`. Explain what went wrong with Ben's RCU-based implementation. How could you fix it?

Answer: Unlike read-write locks, RCU doesn't provide atomicity across updates to different objects. One workaround might be to allocate an entirely new `accs` table during each `transfer()` operation. `accs` could be managed as a single RCU pointer. Another option is to use sequence locks.

VIII Virtualization

For these questions, refer to Adams and Agesen's *A Comparison of Software and Hardware Techniques for x86 Virtualization*i and *Dune: Safe User-level Access to Privileged CPU Features*.

13. [5 points]: Ben is using Baker's algorithm to enable concurrent garbage collection for an application that runs inside a VM. The application primarily accesses a small working set but it frequently allocates new memory from the heap. Assuming Ben is using a VMM with support for both hardware and software virtualization, is Ben better off using Intel's EPT hardware or shadow paging to virtualize the page table? Justify your answer.

Answer: The EPT allows the guest OS to directly manipulate its page table, improving `ptemod` performance dramatically. Since Baker's algorithm requires frequent page table manipulation, it's best to optimize for this case.

14. [5 points]: Recall that VMware's binary translator uses segmentation (and translation for the `%GS` segment) to prevent access to the top of the address space (usually a 4MB region). Normally, most operating systems don't access that region of memory, but suppose a new guest operating system did. Could VMware still support it? Carefully explain your answer.

Answer: Yes, one option would be to trap-and-emulate access to the top region. The VMM could allocate 4MBs of physical memory and then emulate reads and writes to pretend it was being accessed directly.

15. [5 points]: The Dune paper mentions that transitioning a process to Dune mode is irreversible. In a more recent version of Dune, support for reversing Dune mode was added to allow for the reference count on the Dune file descriptor to be dropped, eliminating a memory leak. Suppose you had a process that configured the page table with a very different virtual address-space layout than Linux and Dune are currently providing at the guest-physical layer. What would the process have to ensure before leaving Dune mode to prevent a crash?

Answer: The currently executing code and its stack must be stored in an identity mapped region, such that their guest-virtual address is the same as their guest-physical address.

16. [5 points]: Ben is using Dune to add support for hardware isolation to a web browser. His code includes a security sandbox that uses ring protection to enforce security checks, performing system calls on the behalf of untrusted code. Suppose the untrusted code is executing the following code in ring 3:

```
int foo(int fd)
{
  return read(fd, (void *)0x10000, 0x2000); // read two pages into 0x10000
}
```

A partial listing of the address translations currently programmed into the page table and EPT is as follows:

guest-virtual	guest-physical	host-physical
0x10000	0x20000	0x10000
0x11000	0x31000	0x0A000
0x20000	0x10000	0x1B000
0x21000	0x11000	0x11000

Assuming the security checks pass, provide the read system call(s) that the sandbox would have to perform on the behalf of the untrusted code:

Answer:
read(fd, (void *)0x20000, 0x1000);
read(fd, (void *)0x31000, 0x1000);

IX 6.828

17. [1 points]: What was your favorite topic in 6.828?

Answer:

A. Virtual memory
B. Networking
C. Scalable locks
D. Dune
E. Biscuit
F. Exokernels
G. RCU
H. File System
I. Logging

18. [1 points]: Which 6.828 topic should we discard next year?

Answer:

A. Networking
B. Dune
C. IX
D. Biscuit
E. Binary translation

End of Quiz II — Happy holidays!