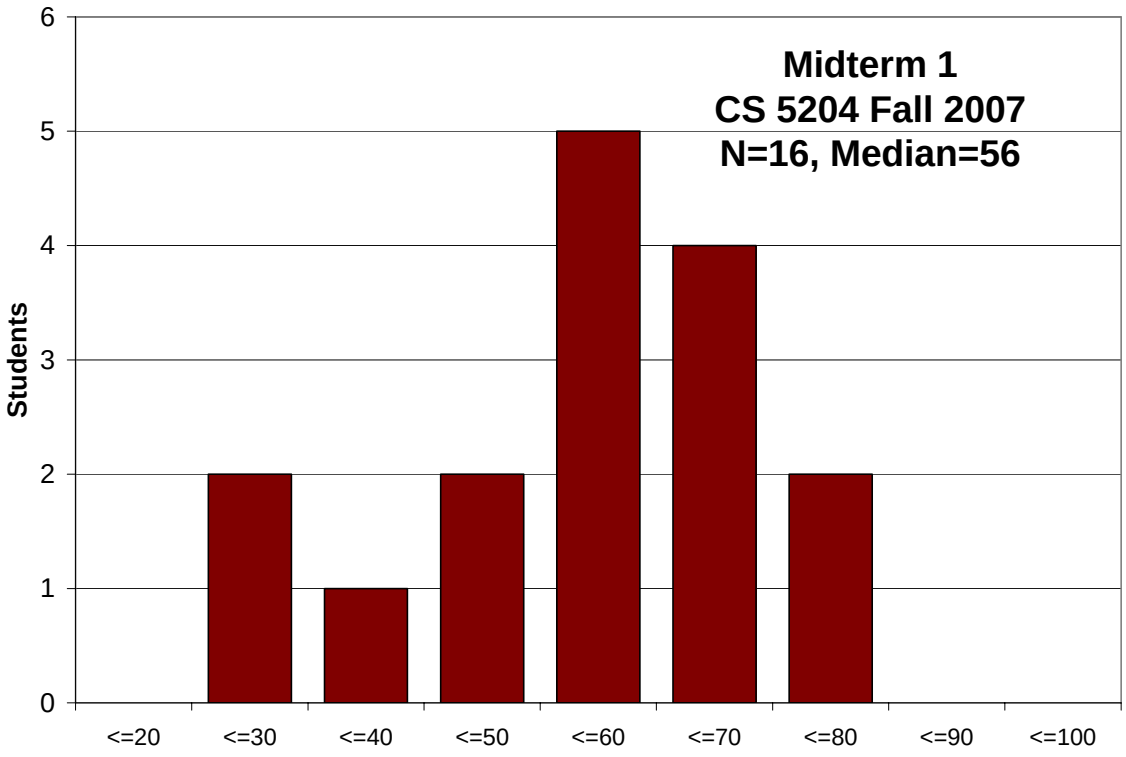
CS 5204 Midterm 1 Solutions

16 Students took the midterm. The table below shows the results for each problem.

Problem	1	2	3	4	Total
Average	2.9	15.9	16.8	19.3	54.8
StDev	3.5	6.7	4.1	9.1	15.2
Median	2	17	17	17	56
Min	0	0	9	0	27
Max	13	24	22	32	79
Possible	20	24	24	32	100

The score distribution is shown below.



1. TAME (20 pts)

a) Consider the following TAME class:

```
class adt {
    bool flag;
    queue<event> equeue;
}

tamed adt::op1(event <> e) {
    if (!flag) {
        flag = true;
        e.trigger();
    } else {
        equeue.push_back(e);
    }
}

void adt::op2() {
    if (equeue.size() == 0) {
        flag = false;
    } else {
        equeue.front().trigger();
        equeue.pop_front();
    }
}
```

i. (10 pts) Choose descriptive names for ‘adt’, ‘op1’, and ‘op2’ that reflect the functionality of this class.

adt = lock
op1 = acquire
op2 = release

ii. (10 pts) Give an example of how it would be used within the TAME framework. (Sketch what call sites for op1 and op2 would look like.)

This example is taken from Section 7.5 in the paper:

```
tamed global_accessor() {
    twait { lock -> acquire (mkevent()); }
    // access critical data while possibly blocking
    lock->release();
}
```

2. Virtual Machine Architectures (24 pts)

At a recent talk at UIUC, Eric Traut, the leader of Microsoft’s Windows Kernel and Virtualization team, introduced the virtualization architecture that will be included in the next version of the Windows operating system, Windows 7.
In his talk, Traut separated existing Type I virtual machine monitors into two groups, comparing them to operating system kernels: one group he referred to as “monolithic hypervisors,” whereas he called the other group “micro-hypervisors.”

a) (2+2+2 pts) What architectural choice motivated him to make this analogy in drawing this distinction? Give an example system for each group!

Monolithic Type I hypervisors include all functionality needed to run guest OS's inside the hypervisor, whereas a "micro-hypervisor" moves some functionality into guest domains. In particular, monolithic hypervisors include device drivers, whereas device drivers are run in guest domains in the other model.
Examples of monolithic hypervisors: IBM/360, VMware ESX, Xen 1.0
Examples of micro hypervisors: Xen 1.2 or later.

Note that it asked for examples of Type I hypervisors, not of monolithic or microkernels in general.

b) (6 pts) Microsoft's hypervisor, he said, will be a "micro-hypervisor." In their model, some guest domains, called "parent partitions," will have direct access to hardware devices; other guests, called "child partitions," must communicate with their parent partition through a software abstraction called a "vmbus" when they wish to access devices.

i. (3 pts) Name one important advantage of this design choice!

It keeps the hypervisor small, making it easier to understand, verify, and likely more robust. Traut said that their hypervisor is well below 100k LOC.
This design choice also means that Microsoft does not have to ask vendors to provide drivers for their hypervisor – the parent partitions will typically run full-blown Windows Server installations. An additional benefit is that child partitions only require a single (possibly enlightened), and likely simple device driver for each device class.
I also accepted other advantages as long as they were related to the design choice of dividing hardware device access/IO between parent domains and child domains, and keeping device I/O out of the hypervisor.

ii. (3 pts) Name one important difficulty MS will have to face in this design!

Judging by Xen's recent performance (discussed in class), it will be difficult to achieve good I/O performance since data may have to be moved across the "vmbus" between parent and child partition, similar to how Xen moves data between front-end drivers in guest domains and backend drivers in domain 0, which hosts the actual device drivers. Although such movement does not always imply copying data, it will involve some form of interdomain/interpartition communication. A second difficulty is in ensuring that a parent partition can access hardware devices safely and efficiently. (Microsoft will rely on architectural additions for virtualized device access for this aspect.)

c) (12 pts) When asked why Microsoft did not follow VMware's path of using adaptive binary instrumentation for CPU and MMU virtualization, he gave two

reasons: enlightenments and upcoming improved hardware support for virtualization.

i. (6 pts) Enlightenments are enhancements to the Windows kernel that make it aware that it is running on top of a virtual machine. For instance, an "enlightened" Windows guest kernel may take one code path when running on bare hardware, and a different path in a virtual machine. Give one example of where enlightenments could be used to benefit virtualization! Explain where the benefit lies.

Two examples may include CPU virtualization and MMU virtualization. For instance, an enlightened kernel may realize that certain instructions will work differently when running de-privileged, and handle that fact. Or, it may avoid certain instructions and call directly into the hypervisor instead. It may know that certain instructions would cause a trap into the hypervisor and avoid frequently executing them – instead, it could ask the hypervisor directly to perform the desired changes to the machine state, in batched form with a single call. Note that there would be benefit even without batching because its easier for the hypervisor to handle a request than having to emulate an instruction.

This strategy would pay off, for instance, when updating its page tables; updates could be batched, and then the hypervisor would be entered only once to install all updates. The hypervisor could also be told directly which physical pages hold page tables.

Recent Linux kernels already include "enlightenments," as part of their VMI implementation. Note that the difference between VMI/enlightenments is different from paravirtualization in that VMI/enlightenments are supposed to be part of the baseline/stock kernel.

ii. (6 pts) AMD's Pacifica architecture includes so-called nested page tables. With a nested page table, the MMU will consult two page tables on a TLB miss: the guest kernel's page table to find the physical address, and the guest domain's page table to translate the physical address to the hardware/machine address.
Explain how nested page tables can simplify the design of a hypervisor for an x86-derived architecture!

Nested page table eliminate the need for the hypervisor to keep shadow page tables that track the guest's primary page tables. (Or, in the case of Xen, the need for paravirtualizing the guest's MMU management.) This will be a huge win, particularly for MMU-intensive workloads (such as workloads that create processes frequently.)
Note that nested page tables don't help with TLB flushes. [TLB flushes are reduced using another architectural addition, tagged TLBs in which each TLB entry is tagged with the guest domain's address space to which it belongs.]

3. VTRR (24 pts)

Linux 2.6.23 contains a new scheduler implementation, called the “Completely Fair Scheduler.” Its developer, Ingo Molnar, describes it as follows[1]:

```
80% of CFS's design can be summed up in a single sentence: CFS
basically models an "ideal, precise multi-tasking CPU" on real
hardware.

"Ideal multi-tasking CPU" is a (non-existent  :-))  CPU that has 100%
physical power and which can run each task at precise equal speed, in
parallel, each at 1/nr_running speed. For example: if there are 2 tasks
running then it runs each at 50% physical power - totally in parallel.

On real hardware, we can run only a single task at once, so while that
one task runs, the other tasks that are waiting for the CPU are at a
disadvantage - the current task gets an unfair amount of CPU time. In
CFS this fairness imbalance is expressed and tracked via the per-task
p->wait_runtime (nanosec-unit) value. "wait_runtime" is the amount of
time the task should now run on the CPU for it to become completely
fair and balanced.

( small detail: on 'ideal' hardware, the p->wait_runtime value would
always be zero - no task would ever get 'out of balance' from the
'ideal' share of CPU time. )

CFS's task picking logic is based on this p->wait_runtime value and it
is thus very simple: it always tries to run the task with the largest
p->wait_runtime value. […]

In practice it works like this: the system runs a task a bit, and when
the task [calls] schedule() (or a scheduler tick happens) the task's
CPU usage is 'accounted for': the (small) time it just spent using the
physical CPU is deducted from p->wait_runtime. [minus the 'fair share'
it would have gotten anyway]. Once p->wait_runtime gets low enough so
that another task becomes the 'leftmost task' of the time-ordered
rbtree it maintains […] then the new leftmost task is picked and the
current task is preempted.

The rq->fair_clock value tracks the 'CPU time a runnable task would
have fairly gotten, had it been runnable during that time'. So by using
rq->fair_clock values we can accurately timestamp and measure the
'expected CPU time' a task should have gotten. All runnable tasks are
sorted in the rbtree by the "rq->fair_clock - p->wait_runtime" key, and
CFS picks the 'leftmost' task and sticks to it.
```

[1] See http://people.redhat.com/mingo/cfs-scheduler/sched-design-CFS.txt

a) (6 pts) Judging only from this description, which of the algorithms discussed in the VTRR paper does CFS resemble?

From this description, CFS sounds like an implementation of WFQ, weighted fair queuing, in which each process is given equal weight.

b) (6 pts) Which terms does the VTRR paper use for what Molnar calls "wait_runtime" and "fair_clock," respectively?

wait_runtime is (the negative of the) service time error, and fair_clock is called queue virtual time.
Note that fair_clock is a single variable per runqueue, not a per process variable. Also note that wait_runtime is not identical to a process's virtual time or virtual finish time (though it is related).

c) (6 pts) CFS's runqueue is implemented using a Red-black tree. Which of the two algorithms (VTRR or CFS) should provide better scalability with respect to the number of tasks or processes?

If indeed CFS is WFQ under a different name, we can expect its scheduling complexity to be O(log N) where N is the number of tasks. Therefore, VTRR, with scheduling complexity O(1), independent of the number of tasks, may provide better scalability.
Usually, when discussing the complexity of a scheduler, we discuss the complexity when making repeated scheduling decisions for a set of ready-to-run tasks. An independent question is how expensive the insertion/deletion of processes into/from the runqueue is. CFS's complexity in this case is O(log N), whereas VTRR's complexity varies: blocked processes can be reinserted in constant time in some cases, in other cases the entire queue must be traversed.

d) (6 pts) Which of the two algorithms (VTRR or CFS) would you expect to provide a smaller service error in the worst case?

If indeed CFS is WFQ under a different name, it would have a bounded service error (see Section 2.4 in VTRR paper) that can never fall below -1 (time slice). By contrast, VTRR's service error may be smaller (larger in terms of absolute value.) Recall that VTRR trades fairness for scheduling efficiency.

4. Drawing Connections (32 pts)

This question asks you draw connections between some of the papers we read. Each sub question is devoted to a different topic.

a) (10 pts) Consider HiStar and the Exokernel. Describe one major aspect in which HiStar's designers followed the design *philosophy* laid out in the Exokernel work.

The Exokernel paper states in Section 3.2 that: "While exokernels allow direct access to low-level resources, exokernel systems must be able to provide UNIX-like protection, including access control on high-level objects where required for security. One of the main challenges in designing exokernels is to find kernel interfaces that allow such higher-level access control without either mandating a particular implementation or hindering application control of hardware resources."

Like the Exokernel, HiStar provides kernel control for the 6 object types it supports, but by allowing threads to create their own categories, the HiStar kernel does not dictate a particular implementation of the security policies used.

b) (10 pts) Consider the proposal made in the OS Support for Virtual Machine paper to introduce a "switchguest" system call, which allows a single process to maintain multiple address space definitions and switch between them. How could this call be integrated into the implementation of an event-based server (such as OkWS)? Name at least one benefit of your proposed design.

Typical event-based servers lack isolation, because all event-handling is done within the same address space. Therefore, if one particular event causes a fault, all subsequent events may be affected, or the entire process may have to be restarted. A multi-address space extension could allow an event-based server to create different address spaces for different "connections" (e.g., by tagging events with a "connection tag"). This would provide similar isolation to using multiple processes, but without the scheduling overhead, and depending on the size of the data that is kept per connection, potentially lower memory overhead. This idea was pursued in HiStar's predecessor, Asbestos.

Some solutions suggested that switching between address spaces may be a faster way to restore an event handler's working state. That's almost certainly not the case, especially on architectures without tagged TLBs. (Activating the new address space is simple – the cost is indirect in the TLB misses that are subsequently encountered.)

c) (12 pts) Consider Eraser and Read-Copy Update Synchronization. Suppose RCU is implemented via "critical_enter()" and "critical_exit()" primitives (as in Figure 5) in the paper, and that quiescent states are flagged explicitly by the application. A possible programming error would be for a process to read some variable within a critical_enter/exit and use the value read during the next entry into that critical section. Could Eraser's Lockset algorithm be *straightforwardly* modified to detect this programming error? Justify your answer!

I would say no, at least not straightforwardly. The lockset algorithm derives information from which locks were held when information was accessed (in a single load/store instruction corresponding to a read or write.) However, detecting

the error described above will require tracing the flow of data between accesses, which is significantly more complex.

Some approaches suggesting flagging shared variables – however, this fails to identify when such shared variables are used after they've been read. Eraser interposes only on accesses (load/stores) to shared variables, not on all uses of derived values. For example, { int local = shared; local2 = local; if (local2) … } The "if (local2)" access is not an event of interest to Eraser. Also, note that the problem asked specifically if Eraser could detect the programming error described in the question, not other errors.