CS 111 Midterm

Jack Zhang

TOTAL POINTS

82 / 100

QUESTION 1

1 1 8 / 8

✓ **- 0 pts Correct**

- 8 pts No answer

- 7 pts Wrong answer

- 4 pts Answer on right track but not correct

- 3 pts Answer needs more detail

QUESTION 2

2 2 7 / 10

- 0 pts Correct

- 10 pts No answer

- 9 pts Wrong answer

- 3 pts Incorrect answers for RR

- 3 pts Incorrect answers for FCFS

- 3 pts Incorrect answers for SJF

✓ **- 3 pts Answer of which has the largest overhead is incorrect or not present**

Round Robin is likely to have the largest overhead due to expensive context switching / being preemptive.

QUESTION 3

3 3 10 / 10

✓ **- 0 pts Correct**

- 10 pts No answer

- 9 pts Wrong answer

- 5 pts Answer on the right track but not correct OR missing part

- 3 pts Answer needs a little more detail OR is slightly off

QUESTION 4

4 4 8 / 8

✓ **- 0 pts Correct**

- 2 pts Miss some details or some sentences are not accurate/correct enough.

- 5 pts Wrote down something, but far from correct/enough.

- 7 pts Wrong answer.

- 7 pts Cannot fully understand/recognize your answer. Please type down your answer using regrading request. Thanks.

- 8 pts No answer.

QUESTION 5

5 5 8 / 8

✓ **- 0 pts Correct**

- 3 pts Didn't explain for shared memory IPC, different processes refer to the exact same page frames or need synchronization.

- 3 pts Didn't explain the copy-on-write property for fork.

- 6 pts Wrong answer or not what we want.

- 7 pts Cannot fully understand/recognize your answer. Please type down your answer using regrading request. Thanks.

- 8 pts No answer.

- 3 pts Missing details.

QUESTION 6

6 6 10 / 10

✓ **- 0 pts Correct**

- 3 pts Didn't consider the case where the page is in RAM.

- 3 pts Didn't consider the case where the page is not in RAM but in disk (page fault).

- 6 pts Wrote down something that makes sense, but didn't cover the main points that we are looking for. For example, didn't answer what operations are required (page table lookup) and didn't cover all

outcomes.

- **9 pts** Cannot fully understand/recognize your answer. Please type down your answer using regrading request. Thanks.
- **10 pts** No answer.
- **3 pts** Missing details.

QUESTION 7

7 7 15 / 15

✓ - **0 pts** **Correct**
- **5 pts** The first 4 iterations are page faults
- **2 pts** Missing last page fault
- **15 pts** Incorrect
- **10 pts** All squares were not filled out
- **5 pts** Incorrect use of the algorithm

QUESTION 8

8 8 15 / 15

✓ - **0 pts** **Correct**
- **15 pts** Incorrect/ Not Done
- **5 pts** Used bit should be set on load
- **5 pts** Page fault on startup
- **5 pts** Incorrect use of the algorithm
- **2 pts** Missing page fault

QUESTION 9

9 16 pts

9.1 a 1 / 4

✓ - **3 pts** **Prolematic**
- **4 pts** Incorrect
- **0 pts** Correct
- **2 pts** Partially correct

We need to support the windows load module and emulate system calls.

9.2 b 0 / 3

✓ - **3 pts** **Incorrect**
- **2 pts** Problematic
- **0 pts** Click here to replace this description.
- **1 pts** Partially correct

A new 2nd level trap handler would be written to intercept the Windows system calls, and pass it on to an emulation layer, which would try to simulate the effects of each Window's system call, using Solaris mechanisms.

9.3 c 0 / 3

- **1 pts** Partially correct.
- **2 pts** Problematic

✓ - **3 pts** **Incorrect**
- **0 pts** Correct

Performance should be okay since user-level instructions don't need to be emulated. Only system calls do.

9.4 d 0 / 3

- **2 pts** Problematic
- **0 pts** Correct

✓ - **3 pts** **Incorrect**
- **1 pts** Partially correct

The architecture does not change!

9.5 e 0 / 3

- **0 pts** Correct

✓ - **3 pts** **Incorrect**
- **2 pts** Click here to replace this description.

Midterm Examination
CS 111, Spring 2019
5/1/2019, 4 – 5:50pm

Name: Jack Zhang Student ID: 004993345

This is a closed book, closed notes test. One single-sided cheat sheet is allowed.

1. What is the benefit of using the copy-on-right optimization when performing a *fork* in the Linux system?

Instead of ~~loading every~~ copying and loading everything at once into the child process, we perform those operations on demand when it is needed. This way, there is not any overhead when spawning a child process through fork. Also saves space, potentially.

and the child process can read from the parent's ages until it actually writes to it, at which point we make a copy.

2. Round Robin, First come First Serve, and Shortest Job First are three scheduling algorithms that can be used to schedule a CPU. What are their advantages and disadvantages? Which one is likely to have the largest overhead? Why?

Round Robin promotes fairness by minimizing response time, thereby boosting interactivity. However, it is terrible for turnaround time. First come first serve, or FIFO is simple to implement but is bad for turnaround time when a time-consuming ~~process~~ job arrives first. It is also not good for response time b/c it does not preempt long jobs for ones that arrive. Shortest job first is better in terms of turnaround time than FIFO, but does not preempt, so is worse than "shortest time to completion". It also isn't good for response time. FIFO is most likely to have the most overhead in both turnaround time and response time if long jobs get queued first, leaving short ones at the end, unable to go until the long ones complete.

3. In a virtual memory system, why is it beneficial to have a dirty bit associated with a page? What are the techniques we can use to reduce the I/O involved in evicting dirty pages?

If a page in memory has been modified, then when it is evicted to the hard disk by the page-replacement ~~daemon~~ daemon, it must actually write to the disk instead of just simply adding to the swap space. This is far more costly, so we use ~~additional~~ dirty bit in the PTE to solve the problem. Building on the LRU approximation clock algorithm, which at first evicted the first PTE w/ use bit = 0, now we give priority to use bit 0 AND dirty bit = 0 (not modified), THEN use bit = 0 and dirty bit = 0. This way, costlier write operations are held off until they must be performed. We can also cluster the dirty pages together, ~~only~~ evicting them and writing to the disk in groups, b/c big writes are faster than many small writes

4. What is the relationship between the concept of working sets and page stealing algorithms?

Working sets work by finding the optimal number of pages to allocate to a process, and evicting if there is too many and page stealing if there is not enough. from another process.

For fork, when the child writes to a page, the VPN of that page no longer points to the PFN that the parent also points to. Instead, it makes a copy, which is now at a different physical address and holds a different value because it was just written to. Shared memory IPC, the VPN's of the threads (In) are constantly mapped to the same PFN, so a change in one thread is reflected in the other

5. Both shared memory IPC and the processes' data areas after a Linux *fork* operation would require the page tables of two processes to point to the same physical page frames. What would be different about the two cases (other than being caused by IPC vs. forking)?

For Shared memory IPC, when one thread/process write to a page, the change reflects for all threads/processes with VPN mapping to that PFN. Thus, there is only one PTE. However, for forks, there is no memory sharing, as a forked process is its own independent process. Thus, each forked process, along with the parent process, has its own PTE with its VPN mapping to the same PFN

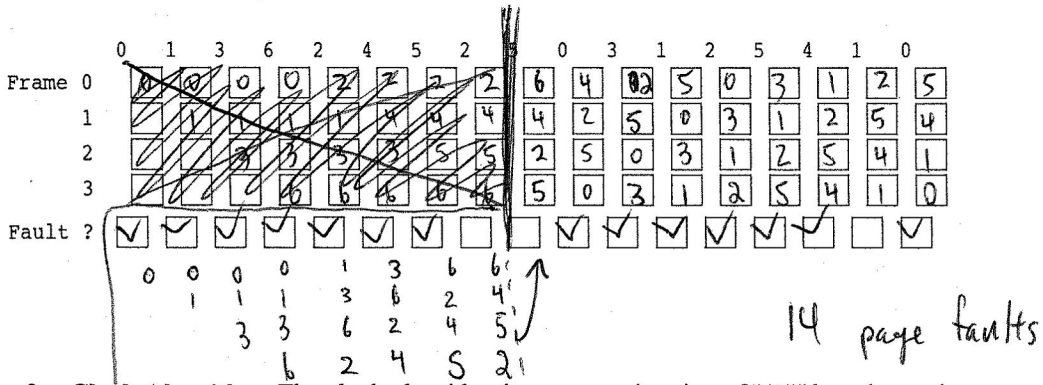
6. In a system using demand paging, what operations are required when a TLB miss occurs? What are the possible outcomes of these operations?

Outcomes:
- ~~TLB~~
- Page in memory
- Page not in memory; on disk
- Page not in memory or disk → segmentation fault

1. TLB fault triggers fault handler, permission level raised to Kernel
2. Searches for the PFN in cache; not found = page fault
3. Page fault handler reads in the page from hard disk at address specified where the PFN usually occupies in the PTE.
4. Page is added to cache
5. Returns to do entire memory reference again → TLB miss, add page to TLB from cache
6. 3rd memory reference = TLB hit.

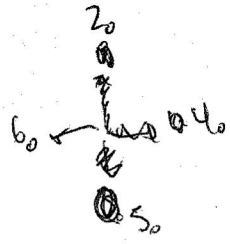
3b. if page not in disk throw error!

7. **Optimal LRU**. Consider the reference string shown along the top of the following graphical structure. The system has four frames. Use the LRU algorithm to select pages for replacement. Place the page number in the proper frame. Mark when page faults occur in the bottom line of boxes. State how many page faults occur. The numbers across the top indicate the reference string.



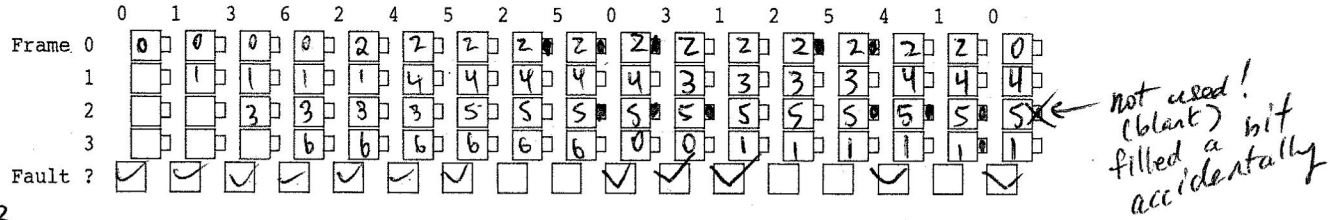
8. **Clock Algorithm**. The clock algorithm is an approximation of LRU based on using one use bit for each page. When a page is used its use bit is set to 1. We also use a pointer to the next victim, which is initialized to the first page/frame. When a page is loaded, it is

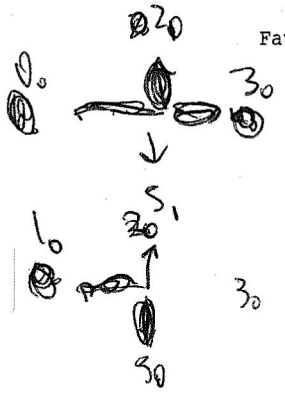
set to point to the next frame. The list of pages is considered as a circular queue. When a page is considered for replacement, the use bit for the next victim page is examined. If it is zero [that page is replaced] otherwise [the use bit is set to zero, the next victim pointer is advanced, and the process repeated until a page is found with a zero use bit].



Consider the reference string shown along the top of the following graphical structure. The system has four frames. Use the clock algorithm described in the previous paragraph. The narrow boxes to the right of the page number boxes can be used to keep up with use bits. Place the page number in the proper frame. Mark when page faults occur in the bottom line of boxes. State how many page faults occur.

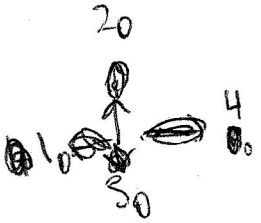
12 page faults





9. In the early 1990s, SUN Microsystems, the maker of the Solaris Operating System, wanted to move from the engineering desktop, where it was well established, to a broader market for personal productivity tools. The best personal productivity tools were all being written for Windows platforms, and SUN was on the wrong side of the applications/demand/volume cycle, which made getting those applications ported to Solaris a non-option.

One approach to their problem was to modify the version of Solaris that ran on x86 processors (the popular hardware platform for Windows) to be able to run Windows binaries without any alterations to those binaries. This would allow Sun to automatically offer all of the great applications that were available for Windows.



(a) What would have to be done to permit Windows binaries to be loaded into memory and executed on a Solaris/x86 system?

machine-level and conventions conform to Windows ABI and make instructions for calling routines consistent w/ those of Windows, although implementation may vary.

(b) What would have to be done to correctly execute the system calls that the Windows programs requested?

Conform to ~~Windows~~ Windows API, modifying, for example, the current Solaris OS implementation of read() to take the same parameters, return the same values, throw the same errors, etc. as ~~Windows~~ Window's read().

(c) How good might the performance of such a system be? Justify your answer.

It depends on the implementation of the routines and subroutines themselves. If they are written to run more efficiently than Windows', then it would have good performance - it wouldn't matter at all that the programs were written for Windows if the new Solaris follows its API and ABI

(d) List another critical thing, besides supporting a new load module format and the basic system calls, that the system would have to be prepared to simulate? How might that be done?

The API must be bound to the ISA

(e) Could a similar approach work on a Solaris/PowerPC or Solaris/SPARC system? Why or why not?

Yes, if the hardware platforms are similar.

Yes, as long as the ISA is written and configured to conform to the Windows' ABI.