CS 111 Midterm

Eric Chen

TOTAL POINTS

76 / 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 **10 / 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

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 **10 / 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 **8 / 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 **2 / 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 **2 / 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

A windows program can directly use memory managed by Solaris.

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: Eric Chen Student ID: 305099495

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?

The benefit of using copy-on-write when forking is that it saves time & space. Take the example of a process with a big data area, copying it to the forked process would take a lot of resources. If we use copy-on-write we only need to perform this expensive task if we're writing to it at some point. Otherwise we don't need to.

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: Advantages are that it's equitable by giving a time slice for each process, prevents starvation. It's wait time is better than fifo & is simply trying to give all processes a chance.
Disadvantage: context switching is huge here

FCFS: Advantage: Simple & easy to use, low overhead b/c no preemption
Disadvantage: If a big process comes first, Avg turnaround time is high.

SJF: Advantage: Turnaround time is good but
Disadvantage: starvation if long process

Round Robin, has largest overhead since it tries to give each process a chance. As a result, the amount of context switching occurs most frequently here relative to SJF & FCFS.

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?

Using the dirty bit with a page is useful because if the page has been written to then when we choose to evict, we know we have to make sure to save the modified contents writing to disk. Otherwise, we'll lose what we changed. One technique we can use to reduce the I/O during eviction is to do the I/O with background threads so that we keep pages clean.

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

working set is the # of pages by process in a time interval while page stealing algo's are an implementation that help maintain the working set. Example, if your set needs more pages you can take one from a different working set. If you overload, then there's thrashing.

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 the process' data areas after a fork, they get copied to the forked process too however their, each process has a different independent data area as its a per-process entity.

For shared Memory IPC after fork remains shareable among processes to read/write and will now include the forked process.

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

Demand paging is bringing pages from disk on demand. When a TLB miss occurs we raise an exception to the trap handler that makes a system call to check the page table for the corresponding page table entry and make sure the present bit is set. If so it is sent to the TLB and reissue the call. If not set, we need to read from disk and put it in the table. So either we find it in the table, bring it from disk, or there's a chance it doesn't even exist.

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.

	0	1	3	6	2	4	5	2	5	0	3	1	2	5	4	1	0
Frame 0	0	0	0	0	2	2	2	2	2	2	2	1	1	1	1	1	1
1		1	1	1	1	4	4	4	4	4	3	3	3	3	4	4	4
2			3	3	3	3	5	5	5	5	5	5	2	2	2	2	0
3				6	6	6	6	6	6	0	0	0	0	5	5	5	5
Fault ?					✓	✓	✓			✓	✓	✓	✓	✓	✓		✓

10 page Faults

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

0

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].

when we load we set use bit to 1 because we asked/need to use it

when page loads, set to point to next frame

it exist dont replace

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.

	0	1	3	6	2	4	5	2	5	0	3	1	2	5	4	1	0
Frame 0	0 ▨	0 ▨	0 ▨	0 ▨	2 ▨	2 ▨	2 ▨	2 ▨	2 ▨	2 ▨	3 ▨	3 ▨	3 ▨	3 ▨	3 ▨	3 □	0 ▨
→ 1	□	1 ▨	1 ▨	1 ▨	1 □	4 ▨	4 ▨	4 ▨	4 ▨	4 ▨	4 □	2 ▨	2 ▨	2 ▨	2 ▨	2 □	2 □
→ 2	□	□	3 ▨	3 ▨	3 □	3 □	5 ▨	5 ▨	5 ▨	5 ▨	5 □	5 □	5 □	5 ▨	5 □	1 ▨	1 ▨
→ 3	□	□	□	6 ▨	6 □	6 □	6 □	6 □	6 □	0 ▨	0 □	0 □	0 □	0 □	4 ▨	4 ▨	4 □
Fault ?	□	□	□	□	☑	☑	☑	□	□	☑	☑	☑	□	□	☑	☑	☑

9 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?

In order for the binaries to be loaded into memory, it must ensure it goes through the stages of being assembly code, to machine language instructions, then through to the linkage editor & then we would load into memory. Going through the build process first is necessary. Lastly, ensure the Interface is as expected once we execute on the system & every task is supported

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

Verifying that the API & ABI is similar and can exhibit same functionality on Solaris as it would on Windows. By ensuring how the hardware utilizes its registers, PC, etc matches the Interface Specification, we can ~~t~~ trust that the outcome will be what we expect. And at the high-level, functions will work as specified in the interface.

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

The performance should still be pretty good since windows is based on x86 and solaris is the same. This suggests the hardware tools that implicitly contribute to performance also are the same for Solaris so it shouldn't be very different.

(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?

Management of memory, we must ensure that when we use Windows that we enforce the goals of transparency, efficiency & protection. This could be done by running processes and making sure performance is not unexpected. If a trivial process consumes incredible amounts of memory there's an issue.

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

It should be able to work on Solaris/PowerPC and Solaris/Sparc system as long as we support the new load module format & basic system calls. If the ISA can support this that is and that the API &ABI is similar so when Windows users use these machines, the interface does not differ.