CS 111 Midterm

Junhong Wang

TOTAL POINTS

88 / 100

QUESTION 1

1 1 **5 / 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**

Needs more detail. How is the performance increased/space saved?

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 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 2 / 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 1 / 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

We would also have to emulate the functionality of the Windows device drivers (specifically displays and printers). This could be extremely complex. It might be easier to provide our own DLLs to directly implement the higher level functionality on a Solaris display server. Other answers to this part could deal with the Windows registry or special Windows networking code.

9.5 e 3 / 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: Junhong Wang Student ID: 504941113

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

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

The benefit is that parent and child processes can share the physical memory, and thus save space.

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?

The advantage of RR is that every process gets fair share of the CPU, and thus good for response time. The disadvantage of RR is that it performs poorly on turnaround time. The advantage of FCFS is that it's simple to implement. The disadvantage is that it performs poorly on turnaround time. The advantage of SJF is that it performs better than FCFS in terms of turnaround time. The disadvantage is that it's still not an optimal algorithm for turnaround time. RR is likely to have largest overhead because it has context switching very frequently.

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?

Dirty bit tells us whether the particular region of code has been modified or not. For example, in paging, when we want to swap a page out to disk, we would like to know whether the page has been modified since it arrived the memory. If it has been modified, we need to write back to disk. Otherwise, we can simply swap the page out, and thus does not involve I/O. Therefore, it's fast. If we want to reduce the I/O even more, we can write dirty page back to disk in background thread, and thus less blocking time when a dirty page is swapped out

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

The idea is that we want to combine LRU and RR to do per-process LRU. Working sets of a process are the pages the process is using.

In page stealing algorithm, processes that need more space for pages will "steal" page allocations from other processes that already have enough working set size.

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)?

In shared memory IPC, the data segment is completely shared.

In forked processes, the data segment is copy-on-write.

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

$$\begin{tikzcd}
\boxed{\text{Start}} \arrow[d] & & \boxed{\text{Load the page table entry to TLB}} \arrow[r] & \boxed{\text{retries instruction}} \arrow[r] & \boxed{\text{TLB hit.}} \\
\boxed{\text{TLB miss}} \arrow[r] & \boxed{\text{Check if the page is in memory}} \arrow[ur, "\text{Yes}"] \arrow[dr, "\text{No}"'] & & & \\
& & \boxed{\text{Load the page from disk to memory}} \arrow[r] & \boxed{\text{retries the instruction}} \arrow[lllu, bend left] &
\end{tikzcd}$$

Note: it will also check whether requested address is valid.

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 ?	✓	✓	✓	✓	✓	✓	✓				✓	✓		✓	✓		✓

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.

	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 ■	4 ■	4 ■	4 ■
1	□	1 ■	1 ■	1 ■	1 □	4 ■	4 ■	4 ■	4 ■	4 ■	4 □	1 ■	1 ■	1 ■	1 □	1 ■	1 ■
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 ?	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓		✓

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?

ABI specifies the executable format for a particular ISA and OS. Thus we could charge the ABI so that Solaris can run Windows binaries.

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

The OS creates trap table at boot time, which maps from system call number to the address of corresponding trap handler within OS. Different OS is likely to have different system call number for the system of a same functionality. (maybe read() is 1 in windows but 3 in Solaris). We need to make sure to translate system-call number of Windows to system-call number of Solaris before looking into the trap table.

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

Worse than running the application on windows OS because there is extra overhead (translation from Windows to Solaris) to be done.

(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?

Solaris should be able to run executables it was able to run before supporting Windows binaries. This can be done by switching ABI depends on extension of the executable files.

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

No, because the hardware architechture of those systems are very different from x86 processors.