CS 111 Midterm

Harrison Paul Cassar

TOTAL POINTS

99 / 100

QUESTION 1

1 Copy-on-write 7 / 8

Click here to replace this description.

✓ **+ 2 pts default - copy on write - parent process's pages not copied for child process**

✓ **+ 2 pts if pages unmodified - they are shared between parent and child process**

✓ **+ 2 pts lazy creation of only modified pages per process**

+ 2 pts unnecessary data copying effort saved(time and memory)

+ 0 pts all wrong

+ 1 Point adjustment

not mentioned memory and time overhead.

QUESTION 2

2 Scheduling 10 / 10

✓ **+ 2 pts largest overhead - round robin**

✓ **+ 2 pts round robin - why largest overhead**

✓ **+ 1 pts round robin - adv**

✓ **+ 1 pts round robin - disadv**

✓ **+ 1 pts fcfs - adv**

✓ **+ 1 pts fcfs - disadv**

✓ **+ 1 pts sjf - adv**

✓ **+ 1 pts sjf - disadv**

+ 0 pts wrong answer

QUESTION 3

3 Running Processes 15 / 15

✓ **+ 3 pts Q1. X is running (A)**

✓ **+ 3 pts Q2. X is running (A)**

Q3.

+ 1.5 pts X is blocked (C) / Y is running (A)

✓ **+ 3 pts X is blocked (C) , Y is running (A)**

Q4.

+ 1 pts X is blocked (C) /Y is running (A) /Z is ready (B)

+ 2 pts Any two right from : X is blocked (C) /Y is running (A) /Z is ready (B)

✓ **+ 3 pts X is blocked (C), Y is running (A) , Z is ready (B)**

Q5.

+ 1 pts X is running (A) /Y is ready (B) /Z is ready (B)

+ 2 pts Any two from : X is running (A)/Y is ready (B) /Z is ready(B)

✓ **+ 3 pts X is running(A) , Y is ready(B) , Z is ready(B)**

QUESTION 4

4 fork() 16 / 16

✓ **- 0 pts (a) 3 times**

✓ **- 0 pts (b) 2**

- 8 pts Incorrect (a)

- 8 pts Incorrect (b)

QUESTION 5

5 Scheduling & turnaround time 18 / 18

✓ **- 0 pts Correct**

FIFO

- 3 pts Order incorrect

- 0.5 pts Turnaround calculation right (value wrong)

- 1.5 pts Turnaround calculation and value wrong

Round Robin

- 3 pts Order incorrect

- 0.5 pts Turnaround calculation right (value wrong)

- 1.5 pts Turnaround calculation and value wrong

SRTF

- 0 pts Order Incorrect --- student argued his misunderstanding was confirmed by the TA. So I gave

3 pts back to him.

- **- 0.5 pts** Turnaround calculation right (value wrong)
- **- 1.5 pts** Turnaround calculation and value wrong
- **- 0.5 pts** A and E messed up

Priority

- **- 3 pts** Order Incorrect
- **- 0.5 pts** Turnaround Calculation right (value wrong)
- **- 1.5 pts** Turnaround calculation and value wrong
- **- 0.5 pts** C and A messed up
- **- 1 pts** Did not find average (Divide by 5)

QUESTION 6

6 Clock Algorithm 16 / 16

- ✓ **- 0 pts Correct**
- **- 16 pts** Incorrect/ Not done
- **- 1 pts** Number of page faults not stated
- **- 4 pts** Not counting faults on startup
- **- 2 pts** 1 Load Error
- **- 4 pts** 2 Load Error
- **- 6 pts** 3 Load error
- **- 8 pts** 4 Load error
- **- 10 pts** 5 Load Error
- **- 12 pts** 6 Load Error
- **- 0 pts** Incomplete table
- **- 8 pts** No use bit
- **- 6 pts** Startup incorrect

QUESTION 7

ABI 17 pts

7.1 Windows & Solaris 4 / 4

- ✓ **+ 4 pts Emulation of Windows / Translator of ABI**
- **+ 2 pts** Mentions the relevance of ABI/OS
- **+ 1 pts** Thought-out reasoning
- **+ 0 pts** Incorrect
- **+ 0.5 pts** Some reasoning

7.2 Executing Programs 4 / 4

- ✓ **+ 4 pts Intercept calls and Simulate**
- **+ 2 pts** Thought-out Reasoning
- **+ 1 pts** Some Reasoning
- **+ 0 pts** Incorrect

7.3 Performance 3 / 3

- ✓ **+ 3 pts Reasonable assessment of tradeoffs**
- **+ 1.5 pts** Attempted assessment of tradeoffs
- **+ 0 pts** No attempt

7.4 Simulation 3 / 3

- ✓ **+ 1.5 pts Identify Problem**
- ✓ **+ 1.5 pts Identify Solution**
- **+ 0 pts** No attempt
- **+ 0 pts** Incorrect / Unclear Proposal

7.5 Solaris and SPARC 3 / 3

- ✓ **+ 3 pts Correct or Thorough + Reasonable Explanation**
- **+ 2 pts** Some flaws but Thorough Explanation
- **+ 1 pts** Flawed but Some explanation
- **+ 0 pts** No attempt
- **+ 0.5 pts** Only yes/no given

X, B,

Midterm Examination
CS 111, Winter 2020
2/5/2020, 2 – 3:50pm

Name: Harrison Cassar Student ID: 505114980

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

1. What is the benefit of using the copy-on-~~right~~ write optimization when performing a *fork* in the Linux system? **8 points**
When performing a fork in the Linux system, the new process created shares the same data segment as its parent process initially. Copy-on-write means that only when either process writes to a data segment page will the page get copied in physical memory, where one process has access to the old data segment page and the other the new copy of the page. The benefit of using this is to potentially save on the overhead of copying the potentially very large data segment in its entirety (all of its pages) until absolutely needed (when one modifies it). Only the pages of the data segment that are written to are copied, not the entire segment.

The changes made to one copy of the data segment are not reflected in the other.

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? **10 points**
Round robin allows for the advantage of majorly minimizing average response time and ensuring fairness between each process for CPU sharing, while having the major disadvantage of increasing average turnaround time immensely and causing many context switches (which produces a lot of overhead). First come first serve (FCFS) allows for the advantage of being very simple to implement, but holds a disadvantage of potentially starving a process of higher priority or shorter length which arrived after a currently running long process. Additionally, FCFS does not specifically exploit anything in relation to execution time, and therefore will have variable delays depending on the order of process arrival. Shortest job first has the advantage of reducing average turnaround time, but has the disadvantage of an increase in average response time, as well as potentially starving a long process of execution time simply because shorter jobs keep arriving before it is able to run. Round robin, because of the significant number of context switches, is likely to have the largest overhead, as each context switch requires the OS to save the current process' state, clean up the CPU, and map and load in a new process and its address space (lot of overhead).

3. Assume you have a system with three processes (X, Y, and Z) and a single CPU. Process X has the highest priority, process Z has the lowest, and Y is in the middle. Assume a priority-based scheduler (i.e., the scheduler runs the highest priority job, performing preemption as necessary). Processes can be in one of five states: RUNNING, READY, BLOCKED, not yet created, or terminated. Given the following cumulative timeline of process behavior, indicate the state the specified process is in AFTER that step, and all preceding steps, have taken place. Assume the scheduler has reacted to the specified workload change. **15 points**

For all questions in this Part, use the following options for each answer:
a. RUNNING
b. READY
c. BLOCKED
d. Process has not been created yet
e. Not enough information to determine

OR None of the above

(a) Process X is loaded into memory and begins; it is the only user-level process in the system. Process X is in which state?

X: RUNNING

(b) Process X calls fork() and creates Process Y. Process X is in which state?

X: RUNNING, Y: READY

(c) The running process issues an I/O request to the disk. Process X is in which state? Process Y is in which state?

X: BLOCKED, Y: RUNNING

(d) The running process calls fork() and creates process Z. Process X is in which state? Process Y is in which state? Process Z is in which state?

X: BLOCKED, Y: RUNNING, Z: READY

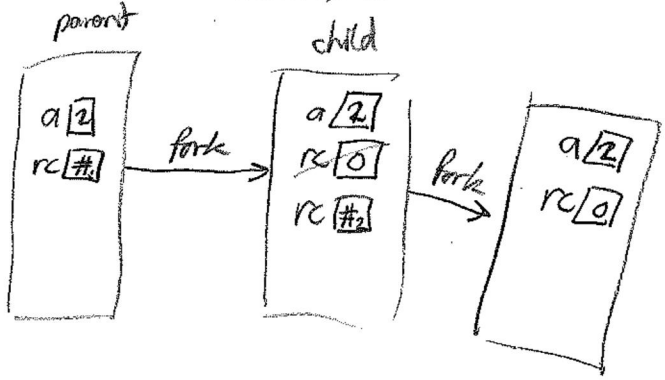
(e) The previously issued I/O request completes. Process X is in which state? Process Y is in which state? Process Z is in which state?

X: RUNNING, Y: READY, Z: READY

4. For the next two questions, assume the following code is compiled and run on a modern linux machine (assume any irrelevant details have been omitted): **16 points**

```
main() {
   int a = 0;
   int rc = fork();
   a++;
   if (rc == 0) { rc = fork(); a++; }
   else { a++; }
   printf("Hello!\n");
   printf("a is %d\n", a);
}
```

child = 0
parent has child PID



(a) Assuming fork() never fails, how many times will the message "Hello!\n" be displayed? 8 points

3 times

(b) What will be the largest value of "a" displayed by the program? 8 points

a = 2, as the data segment of memory will be copied on write (att is a write to memory)

(if race condition applied, it could be a=5, but forking processes and the copy on write functionality prevents this easily)

5. **Scheduling.** Consider the following set of processes, with associated processing times and priorities:

Process Name	Processing Time	Priority
A	4	3
B	1	1
C	2	3
D	1	4
E	4	2

For each scheduling algorithm, fill in the table with the process that is running on the CPU (for time slice-based algorithms, assume a 1 unit time slice). **18 points**

Notes:
- A smaller priority number implies a higher priority.
- For RR and Priority, assume that an arriving thread is run at the beginning of its arrival time, if the scheduling policy allows it.
- All of the processes arrive at time 0 in the order Process A, B, C, D, E.
- Assume the currently running thread is not in the ready queue while it is running.
- Turnaround time is defined as the time a process takes to complete after it arrives
- SRTF, known as shortest remaining time first (SRTF), is another scheduling method that is a preemptive version of shortest job next scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute.

A 11
B 1 ✓
C 11 ✓
D 1 ✓
E 11

Time	FIFO	Round Robin	SRTF	Priority
0	A	A	B	B
1	A	B	D	E
2	A	C	C	E
3	A	D	C	E
4	B	E	A	E
~~4~~ 5	C	A	A	A
6	C	C	A	A
7	D	E	A	A
8	E	A	E	A
9	E	E	E	C
10	E	A	E	C

A=9
B=1
C=11
D=12
E=5

36

A=4 C=7 D=8 E=12

18 20 24 36

A=11 C=7 E=12

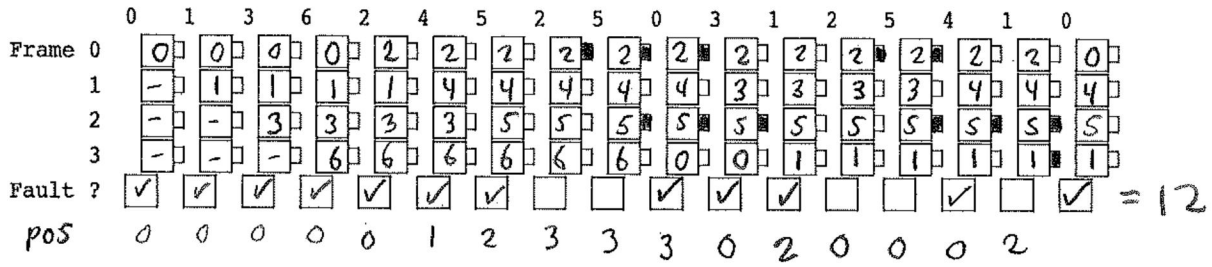
A=8
B=1 C=4
D=2 E=12

17
28
29
38

11	E	E	E	D
Average turnaround time	36/5	36/5	27/5	38/5
	(7.2)	(7.2)	(5.4)	(7.6)

6. **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. **16 points** 12 page faults occur



7. 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. **17 points**

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.

emulate Windows system calls functionalities

(a) What would have to be done to permit Windows binaries to be loaded into memory and executed on a Solaris/x86 system? 4 points

To permit Windows binaries to be loaded and executed on a Solaris/x86 system, the Solaris would need to be able to support the Windows load module in terms of its format and methods/conventions of loading/prepping data for execution as well as be able to emulate Windows system calls and their functionalities (this would be sure to conform to the contract made in the ABI between a Windows/x86 system, which is what these binaries are compatible with).

(b) What would have to be done to correctly execute the system calls that the Windows programs requested? 4 points

To correctly execute the system call requested, the 2nd level trap handler would have to be re-written to ensure that the same operation and result occurs on the Solaris/x86 system as it would on a Windows/x86 system. Essentially, the binaries should not be able to tell the difference.

(c) How good might the performance of such a system be? Justify your answer. 3 points

Likely, at an overall sense, the performance would be largely unaffected, as this modification is made only for system calls, which invokes the OS and its privileged kernel mode of operation. Since the vast majority of execution will be done in user mode (limited direct execution), system calls would not be invoked, and therefore performance remains largely unchanged. compared to the Windows/x86 system. Essentially, a little worse, but largely unaffected.

(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? 3 points

Another potential thing would be that the system would have to work with a vastly different workload, simulating the potentially more interactive-ness of a Windows system. This simulation might be achieved by altering the scheduling algorithm that is utilized to schedule processes on the CPU for execution (in the example given potentially switching to a scheduling algorithm that more favors a decrease in average response time, such as round robin).

(e) Could a similar approach work on a Solaris/PowerPC or Solaris/SPARC system? Why or why not? 3 points

No, a similar approach could not, as a complete alteration of the architecture (and its ISA) means that the system's ABI (or contract between the OS, binaries, and ISA) is completely different than what the Windows binaries were compiled for/are compatible with. The OS developers of the Solaris OS do not have the ability to also simulate an x86 architecture as well (at least not effectively, as then the performance would significantly decrease because of the necessity of the Windows binaries to, on every instruction, even those that do not require system calls, invoke the OS). This performance draw would be already enough to make this approach unusable and unmarketable.