Operating System Concepts, Midterm Exam

November 23, 2020

1. (10pt) Answer the following questions on privileged instructions and OS protections.
   (a). (3pt) Explain why the design of kernel mode and the privileged instructions can allow OS to protect a computing system?
   Ans: The operation of a computer system is protected by the privileged instructions.
   Privileged instruction can only be executed in kernel mode.
   So if a user want to execute privileged instructions, he **must enter the kernel mode** by **calling system calls** to execute these instruction on behavior of the user.
   Therefore, the OS can have the control to protect the system.
   Grading: -2pt, if only mention Privileged instruction can only be executed in kernel mode.
   -2pt, if not mention what kind of actions or instructions should be defined as privileged instructions.
   Wrong answer:
   - Simply say that the privileged instructions avoid users to execute certain instructions or modify kernel.
     The purpose of privileged instructions is not to deny users from executing them, but to ask users to execute them through kernel. A program can execute privileged instructions, just have to go through kernel/system calls.

   (b). (3pt) Why we cannot fully protect IO devices by making all the IO instructions as privileged instructions?
   Ans: If the memory is not protected, the hacker can still overwrite the IO data and instruction.
   (c). (2pt) As we known, memory access instructions are NOT privileged instructions. What would be the main problem if we make them privileged instructions?
   Ans: every memory access instruction has to trigger system call. It will be time consuming and cause performance degradation.
   (d). (2pt) Consider an instruction that is used to disable interrupt, should it be a privileged instruction? **Explain why.**
   Ans: Yes, because it can affect other processes.

2. (5pt) (a) (2pt) What is POSIX interface? (b) (3pt) Why a program only needs to re-compiled, but not re-written when running across different OS with the same POSIX interface?
   Ans:
   (a) POSIX is the standard API interface defined by a class of OS.
   (b) Because the OS have the same API, the user code only needs to be re-compiled.

3. (6pt) Answer the following questions for comparing the shared memory and message passing communication models.
   (a). (4pt) When system calls will be required during the use of these two communication methods?
   Ans: (2pt each) share segment initialization or do synchronization for shared memory ; send/recv for message passing
   Wrong answer:
   shared memory: Simply say access memory
   (2pt) If the size of the data to be exchanged is large, which communication model can provide better performance? Explain why.
   Ans: Because the memory copy time grows proportional to the data size.

4. (10pt) Answer the following questions on the three thread models: one-to-one, many-to-one, and many-to-many.
   (a). (3pt) Give **one** limitation or drawback from using **each of the three models**.
   Ans: 1pt each

one-to-one: only limited number of kernel threads are available
many-to-one: all the user threads are blocked by a single "kernel thread"
- Wrong answer: one "user thread" blocks, all the other user threads blocks

many-to-many: more complex implement, and thus can introduce more runtime overhead

(b). (2pt) Why creating a user thread is faster than a kernel thread?
Ans: it doesn't involve system calls.

(c). (2pt) Does it require context switch when switching the execution among user threads? **Explain your answer.**
Ans: No, context switch only needs when we switch kernel threads.

(d). (3pt) Describe a **scenario in which** using many-to-one thread model can result in higher CPU utilization than using one-to-one thread model?
Ans: When there is only one core, and the program is CPU-bound without IO.
Any other reasonable scenario can be correct as well.
-1.5: Only mention one of {single core, CPU-bound without IO}
-3: incorrect explanation
-3: scenario with "similar" utilization (the question ask for 'higher' utilization)

5. (8pt) (a) (2pt) A library can be compiled into static or dynamic library. Will the size of executable binary file from compilation for the dynamic linking library be smaller than the static linking library? **Explain why.** (b) (3pt) When using dynamic linking technique, what does the **compiler** and **OS runtime** has to do for running our program? (c) (3pt) If we compile our program on one machine and run on another machine, **static linking** can be used to avoid the "**library not found**" error at runtime. **Explain why**.
Ans:
(a) Yes, because the linked library is not included in the binary file.
(b) (1.5pt each) Compiler must insert the stub (or the code that connect user program to library); Runtime must find the existing library in memory. The library must be loaded and linked at runtime if not found.
(c) Because the library will be included in the binary file already by static linking.

Wrong answer:
- (a) Only mention avoiding duplicate libraries, which save "memory", but do not mention why "binary file" is smaller.
- Mix-up dynamic linking with dynamic loading
- Program vs. process

6. (8pt) Address binding can happen at compile-time, load-time, and run-time. Answer the following questions.
(a). (3pt) In what kind of computer system or what use scenario, will compile-time binding become the best option. **Explain your answer.**
Ans: Simple computer systems where no dynamic memory management (page sharing, virtual memory) is needed to be supported.

(b). (2pt) If we want to support virtual memory in a load-time address binding system, what restrictions do we have during page swapping?
Ans: The program memory must be swapped back to the same physical memory location.

(c). (3pt) Is it possible to implement dynamic linking with load-time address binding? **Explain why**.
Ans: No, because the library linking address must be determined at runtime.

7. (9pt) Answer a larger page size will increase, decrease or say the same for the following metrics? Briefly **explain each of your answers**.
(a). (1.5pt) The page fault rate.
Decrease, because each fault will bring more data back to memory

(b). (1.5pt) The page fault handling time.
Increase, because more data needs longer data transfer time.

(c). (1.5pt) The TLB miss rate.

Decrease, because each page bring more data back to memory

(d). (1.5pt) The memory utilization.

Decrease, because more internal fragmentation.

(e). (1.5pt) The number of page table entries. (Assume the logical address bits remain the same.)

Decrease, because the page offset bits become more.

(f). (1.5pt) The logical address space that can be allocated to a process. (Assume the logical address bits remain the same.)

Stay the same, because the total logical address bits remain the same.

8. (14pt) Answer the following questions about page replacement algorithms

(a). (2pt) Why LRU can often achieve a pretty good page fault rate in practice?

Because program has locality, the page recently used is likely to be used again soon.

Wrong answer: LRU is an approximation of the optimal algorithm

(b). (4pt) Explain why stack algorithm can never cause Belady's anomaly.

Stack algorithm means the page stored with N frame is a subset of the page stored with N+1 frame.

Therefore, all the page that are hits with N frame will still be hits with N+1 frame.

Hence, the number of page faults with N+1 framework will not be more that the result with N frame.

(c). (4pt) The optimal page replacement algorithm needs to know the future page references. We know some compiler technique can help to analyze the memory reference from instructions. But it is still difficult to get the accurate references. **Give two reasons** to explain it**.**

i. The code path can be dynamically change (like branch).

ii. The memory can be dynamically allocated.

iii. Exception or error at runtime

Wrong answer:

- **due to change of physical address or frame**
- **due to overhead. (compiling doesn't cause runtime overhead.)**
- **stack algorithm doesn't care the stored pages are optimal or not**

(d). (4pt) Considering the following program, assume N is very large number, and the computer only has two frames available. How can you rewrite the program to minimize page faults?

```
for (int i = 0; i < N/2; i++)
      A[i] = B[i] + B[N/2+i];
```

```
      A[i] = B[i];
for (int i = 0; i < N/2; i++)
      A[i] += B[N/2+i];
```

9. (6pt) You've been hired by Orange Computer to help design a new processor and Orange Pro laptop. After choosing the display, case, and other components, you are left with some money to spend on the following components:

Item	Latency	Minimum Size	Cost
TLB	10ns	256 entries	$0.1 / entry
Main memory	180ns	2 GB	$10 / GB
Magnetic disk	8ms (8M ns)	300 GB	$0.1 / GB

The page size is fixed at 64 KB. Assume you want to run up to 20 IO-bound applications simultaneously. Each application has an overall maximum memory size of 1 GB and a working set size of 256 MB. TLB entries do not have Process Identifiers. Answer the following questions:

(a). (2pt) Which component will you consider to spend money on buying first? Explain why.

Ans: disk, because it is slowest component, and it will become the performance bottleneck.

(b). (4pt) If you only have $420 left to buy RAM and TLB, how much will you spend on each of them. **Explain why**.

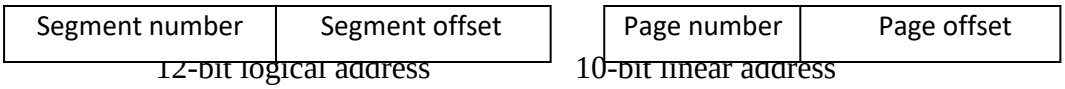
First, we need to have enough memory to avoid page fault□256MB*20=5GB□ $50

Then we need enough TLB entries to cover the working set of a process. □ 256MB/64KB = 4096 □ $409.6

Since we only have $420, we should choose RAM over TLB.
 Ans: $50 on RAM, $370 on TLB.
*grading: -2: if ~$410 for TLB, $10 for RAM. All other answer will not receive any point.

10. (9pt) Answer the following questions on the three page table structures: hierarchical, hashed, inverted.
(a). (3pt) Why we have to use these page table structures when the size of page table grows too large?
Because consecutive pages can be placed on dis-contiguous frames.
MMU will need reference pointer to find pages on different frames.
*grading: as long as the problem of finding pages spread on frames is mentioned can be consider as correct answer.
(b). (3pt) Why hashed page table structure likely to have faster memory translation time than the hierarchical page table structure, when the actual user program is small and maximum logical address space is huge.
Hashed page table doesn't need to store the invalid/unallocated table entries. So the
(c). (1.5pt) Why inverted page table often take longer address translation time than the other two.
It needs to search through the frame table to find the page number
(d). (1.5pt) Why inverted page table requires less memory space to store its page table than the other two.
It only requires a frame table which is used for all the process, while other structures have to create one page table per process.

11. (15pt) Consider the case of translating a **12-bit hexadecimal logical address** "27A" into **10-bit physical address** with a **paged segmentation** (i.e., first segmentation, then paging) memory management scheme on a byte addressing machine. Given the segmentation and page table below. Answer the following questions.



Segmentation table

Page table

(a) (2pt) What is the page size? 2^(10-2)=256 Bytes
(b) (1.5pt) Which segmentation table entry is used in this translation? (entry index starts from 0)
Physical address: 0010 0111 1010
Segment offsets: 10 bits; Segment number: 2 bits
Segment entry: 0
(c) (1.5pt) What is the translated linear address? (**in binary format**)
First check offset: 11 0000 0000 > 10 0111 1010
Linear address: 00 0010 0000 + 10 0111 1010 = 10 1001 1010
(d) (1.5pt) Which page table entry is used in this translation? (entry indexes start from 0)
Page number: 2 bits ; page offset: 8 bits
Page table entry: 2
(e) (1.5pt) What is the translated physical address? (**in binary format**)
Physical address: 11 1001 1010
(f) (3pt) Give **an example of logical address** that will cause segmentation fault, and **explain** how it causes the segmentation fault.
Must satisfy the three requirements below
(i) the logical address must be 12 bits

(ii) must explain which segment entry is used
(iii) the segment offset must be larger than the segment limit from the segment table entry

(g) (4pt) Assume a TLB is used for translation from page number to framework, and page 0 and 1 already cached in TLB. The TLB access time is 10ns, the memory access time is 100ns, the disk access (page fault handling) time is 10,000ns. What will be the end-to-end effective memory access time (including address translation and data reading) for the "27A" memory access in this question? **Must explain your equation for calculation.**

Ans: 100 (segment table access) + 10 (TLB miss) + 100 (page table access) + 10000 (page fault handling) + 100 (segment table access) + 10 (TLB hit) + 100 (data access) = 10420

-1: if segment table access time is missing
-1: if page table access time is missing
-2: instruction restart time is missing
-1: data access time is missing
-1: misjudge the TLB hit/miss in the first instruction execution
-1: misjudge the TLB hit/miss in the second instruction execution