Ph.D. Qualifiers Exam Fall 2019
Operating Systems

Computer Science Department,
New Mexico State University

Exam time: 120 min. The exam contains a total of 7 problems, all equal weight. Choose and solve any 5 out of 7.

If you believe that you cannot answer a question without making some assumptions, state those assumptions in your answer.

Irrelevant verbosity will not gain you points. Clear and crisp answers will be appreciated.

This is a closed-book, closed-note exam.

Textbook:
Operating Systems Concepts, Ninth Edition, by Silberschatz, Galvin, and Gagne published by John Wiley and Sons.

Qn1. There are two major design types for operating system kernels: monolithic kernels and microkernels. Which design better satisfies the following requirements, monolithic kernel, microkernels or both? Justify your answers.

1. Convenient access to operating system data structures by the kernel-level process.
2. Adding/modifying operating system components by kernel developers.
3. Strong security and reliability.

1. **Monolithic – access from any part of the kernel to any other part is possible. With a microkernel, access is via message passing.**
2. **Microkernel – no (or little) modification is required in the kernel as the service is in user space.**
3. **Microkernel – services are isolated from each other. If one service crashes it does not directly affect others.**

Textbook source: Chapter 2.7 from page 78 to page 83

Qn2. Process synchronization:

1. Assume you have a system with a static priority CPU scheduler. Assume the scheduler supports preemption. Describe what the program below prints if the priorities are set such that T2 has a high priority, T1 has the middle priority, and T0 has the low priority. Assume the system starts with only T0 executing. Assume the semaphore *mutex* is initialized to 1.

Void T0(){
printf("T0-S\n");
wait(mutex);
printf("T0-1\n");
StartThread(T1);
printf("T0-2\n");
signal(mutex);
printf("T0-E\n");
}
	Void T1(){
	printf("T1-S\n");
	StartThread(T2);
	printf("T1-E\n");
	}
		Void T2(){
		printf("T2-S\n");
		wait(mutex);
		printf("T2-1\n");
		signal(mutex);
		printf("T2-E\n");
		}

2. What changes, if any, would happen to the order of the printfs if priority donation were added to the CPU scheduler? Priority donation: if a higher priority thread A is waiting on a mutex held by a lower priority thread B, thread A donates its priority to thread B. When thread B releases the mutex, its priority reverts to its original value.

1. T0-S, T0-1, T1-S, T2-S, T1-E, T0-2, T2-1, T2-E, T0-E

2. T0-S, T0-1, T1-S, T2-S, T0-2, T2-1, T2-E, T1-E, T0-E

Textbook source: Chapter 5.6 from page 214 to 216; Chapter 6.3.3 from page 270 to 271

Qn3. Scheduling:

1. Compare and contrast Round Robin scheduling with Shortest Job First (SJF or SRTF) scheduling. Briefly discuss the strengths and weaknesses of each scheme with respect to the usual goals of a CPU scheduler. Why do most modern CPU schedulers combine Round Robin and SJF by giving CPU priority to I/O bound jobs?

2. What is priority inversion, and why is it bad? Illustrate with an example. Use your example to illustrate one technique that avoids priority inversion.

1. SJF: short processes are executed first and then followed by longer processes.

- **Advantage: minimum average waiting time.**
- **Disadvantage: 1) the time taken by a process must be known by the CPU beforehand. 2) Longer processes may suffer starvation.**

Round Robin: each process is served by the CPU for a fixed time quantum.

- **Advantage: 1) starvation free. No process is left behind. 2) better response time.**
- **Disadvantage: 1) the throughput largely depends on the choice of the time quantum. 2) long average turnaround time.**

By combining Round Robin and SJF, the scheduler can divide processes into different classes and assign different algorithms according to their scheduling needs. As I/O bound jobs are usually interactive processes and have shorter CPU burst, they should be given high priority to gain better responsive time.

2. Priority inversion comes under priority-based scheduling. It is a scenario where a high priority task is indirectly preempted by a lower priority task.

The problem can be solved by priority-inheritance protocol. All processes accessing resources needed by a higher-priority process inherit the higher priority until they are finished with the resources.

Textbook source: Chapter 5.6.4 from page 217 to 218; Chapter 6.3 from page 266 to 276

Qn4. Consider a simple paging system with the following parameters: 2^{32} bytes of physical memory; page size of 2^{10} bytes; 2^{16} pages of virtual address space.

1. How many bits are in a virtual address?
2. How many bytes in a frame?
3. How many bits in the physical address specify the frame?
4. How many entries in the page table?
5. How many bits in each page table entry? Assume each page entry includes a valid/invalid bit and padding bits to make its size a power of 2.
6. What is the effect on the page table if the physical memory space is reduced by half?

1. 10+16=26 bits

2. 2^{10} **bytes**

3. 22 bits

4. 2^{16} **entries**

5. 32 bits including 22 bits to specify physical frame, 1 valid/invalid bit, 9 padding bits

6. 32 bits including 21 bits to specify physical frame, 1 valid/invalid bit, 10 padding bits

Textbook source: Chapter 8.5 from page 367 to371

Qn5. A process has four frames allocated to it. The time of the last loading of a page into each frame (i.e., Time Loaded), the last access to the page in each frame (i.e., Time Referenced), and the virtual page number in each frame are as shown below (the times are in clock ticks from the process start at time 0 to the event).

Virtual Page Number	Frame	Time Loaded	Time Referenced
20	0	60	161
22	1	130	160
24	2	10	162
31	3	20	163

A page fault to virtual page 23 has occurred at time 164. Which frame will have its contents replaced for each of the following memory management policies?

- FIFO (first-in-first-out)

  Frame 2, page 24 will be replaced

- LRU (least recently used)

  Frame 1, page 22 will be replaced

- Optimal. Use the following reference string: 23, 25, 23, 25, 20, 22, 31, 23, 24.

  Frame 2, page 24 will be replaced

Textbook source: Chapter 9.4 from page 413 to 416

Qn6. Consider the following program.

```
#define N 64
int A[N, N], B[N, N], C[N, N];
int i, j;

for (j = 0; j < N; j ++)
      for (i = 0; i < N; i++)
            C[i, j] = A[i, j] + B[i, j];
```

Assume that the program is running on a system using on demand paging and the page size is 4KB. Each integer is 4 bytes long. It is clear that each array requires 4-page space. As an example, the following elements from matrix A will fit in one page: A[0,0]-A[0,63], A[1,0]-A[1,63], A[2,0]-A[2,63], ... to A[15,0]-A[15,63]. A similar storage pattern can be derived for the rest of array A and for arrays B and C.

Assume that the system allocates a 4-page working set for this process. One of the pages will be used by the program and three pages can be used for the data. Also, assume that two index registers are assigned for i and j (so, no memory accesses are needed for references to these two variables).

1. How many page faults does this program generate?
2. How would you modify the program to minimize the page fault frequency? What will be the frequency of page faults after your modification?

1. $3 \times 4 \times 64 = 768$ **page faults**

2. Change the loop order as

```
#define N 64
int A[N, N], B[N, N], C[N, N];
int i, j;

for (i = 0; j < N; j ++)
      for (j = 0; i < N; i++)
            C[i, j] = A[i, j] + B[i, j];
```

In this way, we only have 3×4 **pages faults, the frequency of page faults is** $\frac{12}{3 \times 64 \times 64}$

Textbook source: Chapter 9.9.5 from page 442 to 443

Qn7. Apple originally did not include support for multi-tasking in its iphone OS. Why do you think they made this design decision? The newer iphone OS, however, includes support for multi-tasking. What were the technical arguments that might have prompted Apple engineers to make this change? What (if any) problems might this give to the user of the device?

- **It's an open question.The original IOS may be limited by its memory size, CPU capability or battery life. Single-tasking design is more reliable and effective. The newer iphone comes with larger memory and higher computational ability, which is able to support sophisticated OS structure.**
- **To support multi-tasking, system may need large memory, sophisticated CPU schedulers, complex frame allocation and page replacement algorithms, protection mechanisms etc. to efficiently manage all processes and avoid possible deadlock and thrashing.**

Textbook source: Chapter 3.2 from page 111 to 115