Homework 4

Chenggang Wang — M12645906

October 2019

1. Consider the following snapshot of a system:

	Allocation	*Max*	*Available*
	A B C D	*A B C D*	*A B C D*
P_0	2001	4212	3321
P_1	3121	5252	
P_2	2103	2316	
P_3	1312	1424	
P_4	1432	3665	

Answer the following questions using the banker's algorithm:

a) Illustrate that the system is in a safe state by demonstrating an order in which the processes may complete.

b) If a request from process P1 arrives for (1, 1, 0, 0), can the request be granted immediately?

c) If a request from process P4 arrives for (0, 0, 2, 0), can the request be granted immediately?

Answer:

a) Need matrix is:

A	B	C	D
2	2	1	1
2	1	3	1
0	2	1	3
0	1	1	2
2	2	3	3

So, if the system execute the processes in order P_0, P_1, P_2, P_3, P_4, then the system will be in a safe state

b) Yes, the request can be granted immediately, because that available resources is larger than the request resources and once it done, it will release the resources and then the system is still in safe state

c) Same as the above question, the available resources is larger than the requested resources and system is still in safe state, so it can be granted immediately.

2. Compare the memory organization schemes of contiguous memory allocation, pure segmentation, and pure paging with respect to the following issues:

a) External fragmentation

b) Internal fragmentation

c) Ability to share code across processes

Answer:

a) **External fragmentation:** Contiguous allocation with fixed-size partitions will not have any issue with external fragmentation, however contiguous allocation with variable sized partitions does. Pure paging is fixed-size partition scheme, so it does not suffer from external fragmentation, while segmentation does suffer from external fragmentation because it is variable sized partition scheme.

b) **Internal fragmentation:** When allocate memory with segmentation and variable-sized partitions methods, it will be assigned exactly as large as it needs to be. Meanwhile, contiguous allocation with fixed-size partitions and paging will have issues of internal fragmentation when partitions and pages are not completely filled.

c) **Ability to share code across processes:** Contiguous allocation provides no support for code sharing. In segmentation, as long as the segments of a process do not mix text and data, we can easily share code between processes. We simply adjust the segment tables of each process to point to the same segment of code in memory. For security reasons, however, it would probably be desirable to have some method of preventing processes from modifying code, since doing so would allow one process to change the code executed by another. In pure paging, code can be shared across processes simply by sharing page frames. To do this, adjust the page tables of the two processes so that their different logical pages map to the same physical page frame. However, we do need to make certain that no page frame contains any data, which should not be shared. We could accomplish this by, for example, padding the last page of the text segment of the process with no-op machine language instructions.

3. Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.

a) How many bits are required in the logical address?

b) How many bits are required in the physical address?

Answer:

a) Since that logical address space have 256 pages with a 4-KB page size, so total logical space is 1024-KB, so we need 20 bits for the logical address.

b) Since that physical memory have 64 frames, so total physical memory space is 256-KB, so we need 18 bits for the physical address.

4. Consider a paging system with the page table stored in memory.

a) If a memory reference takes 50 nanoseconds, how long does a paged memory reference take?

b) If we add TLBs, and 75 percent of all page-table references are found in the TLBs, what is the effective memory reference time? (Assume that finding a page-table entry in the TLBs takes 2 nanoseconds, if the entry is present.)

Answer:

a) 100 nanoseconds, 50 ns to access the page table, 50 ns to access the word in memory

b) TLB is short of Translation Lookaside Buffer. The effective memory is 64, 75% of the time to access the table is 2 ns, 25% of the time to access is 50 ns, so in total is: (0.75 * 2 + 0.25 * 50) + 50 = 64.

5. A simplified view of thread states is Ready, Running, and Blocked, where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (for example, waiting for I/O). This is illustrated in Fig. 1. Assuming a thread is in the Running state, answer the following questions, and explain your answer:

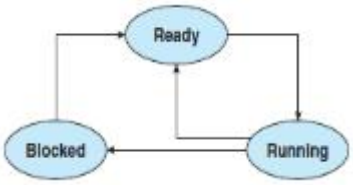
a) Will the thread change state if it incurs a page fault? If so, to what new state?

b) Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what new state?

c) Will the thread change state if an address reference is resolved in the page table? If so, to what new state?

Answer:

a) Yes, it will change state, it will change to from running state to blocked state.
When a page fault occurs, the process starts to wait for I/O operation



to finish. The OS will perform following operations while the process is waiting: The OS will firstly check if the page is really invalid or just on disk. If the page is on disk, then the OS will try to find a free memory frame, schedules a disk access to load the page into the frame, updates the page table with the new logical-physical mapping and the valid bit of that entry, In the end, the OS will schedule to execute that process again and restarts the process by change its state from Blocked to Ready.

b) It depends. If TLB miss occurs, the page number is used to index and process the page table. If the page is already in main memory, then TLB is updated to include the new page entry, while the process execution continues since there is no I/O opeartion needed. If the page is not in the main memory, a page fault is generated, in this case, the process needs to change to the **Blocked** state and wait for I/O to access the disk.

c) No, it will not. Because once the address reference is resolved in the page table, it means that the page needed is already loaded in the main memory already, so no more I/O operation is needed.

6. Programming problem: