2024 Spring Semester

Operating System Final Exam (40pts)

Deadlock (10pts)

1. Assume that a computer system currently runs five processes which are in the running state. Each process is single-threaded and provided its maximum resource requirements in its creation. Below is a resource allocation table taken for the system:

Process ID	Allocation	Maximum	Available
	A B C	A B C	A B C
P1	1 1 2	3 3 3	2 1 2
P2	1 2 2	3 2 5	
P3	1 2 1	1 3 1	
P4	0 1 0	3 1 5	
P5	0 0 0	4 3 0	

a. Describe validity of the processes first, and then explain which state the current system is by using Banker's algorithm. (3pts)

-> First of all, Banker's algorithm detects the unsafe state, not the deadlock state.

The very first job to do is checking whether the processes requests lower or equal to the maximum system resource instances. By adding the allocated and available resource instances, the system has 5, 7, and 7 instances for A, B, and C resources. All processes' maximum instances are less than them, so the processes seem valid.

Second, by applying the Banker's algorithm, let's consider index starts from 1 in each iteration. Then, the sequence for safety check exists as <P3 (2,2,2), P1 (3,3,4), P2 (4,5,6), P4 (5,7,7), P5 (5,7,7)>, which means the system is in the safe state.

b. If P5 requests two resource instances of C, explain how to handle this request as if you were an OS of this system. Also, explain whether granting this request would make the system be deadlocked, too. (3pts)

-> P5 claimed no resource instances at its creation, therefore this request is definitely invalid so it should not be granted.

However, if we apply the deadlock detection algorithm for the single request of 0, 0, 2 by P5, the sequence of <P3, P1, P2, P4, P5> still does not change, therefore the system becomes not deadlocked though the request is invalid.

c. In the deadlock prevention, OS implements its structure to inherently block one of four deadlock requirements. Please list up the four requirements and explain how to block each

requirement with its side effect in the general OS. (4pts)

-> Mutual Exclusion: making all OS components be sharable could violate this requirement, but some components in the general OS, e.g. mutex lock, are intrinsically nonsharable, which requires mutual exclusion.

Hold and Wait: executing processes if and only if all required resource instances are allocated. This method definitely blocks the deadlock, however, resource utilization becomes low and starvation possibly occurs.

No Preemption: Enabling a process steals resource instances from the other process holding the required resources. Same to the Hold and Wait case, starvation can occur if the victim selection is not adequately designed.

Circular Wait: Numbering all resources and granting resource requests in one direction only, not to make any cycles in resource requests. This restricts all processes' behavior to acquire resources in the fixed order, therefore reduces overall efficiency.

2. Main Memory Management (10pts)

a. Please answer with **True or False** if you know the answer. (5pts, **right answer 1pt, blank 0pt, wrong answer -1pt**) (3pts)

(i) If a process repeatedly calls for the system libraries which are dynamically linked, then the stub finds and runs the relevant codes.

-> False, the first call lets the stub find the relevant address, but the indirection table is updated during this first call and subsequent calls do not require the stub.

(ii) Paging is a memory management technique that divides both physical memory and logical address space of a process into fixed-size blocks.

-> True

(iii) By using the paging with segmentation, OS can run a process that exceeds the physical main memory size.

-> False. Virtual memory enables this functionality.

(iv) Demand paging requires a whole process image to be uploaded in the main memory for running.

-> False. It just uploads pages which is referenced by a process.

(v) Increasing the page size also increases the page table size.

-> False. Increasing the page size for a system decreases the number of page table entry so the page table size decreases.

b. Assume that a system implemented a three-level hierarchical paging system. Here, the

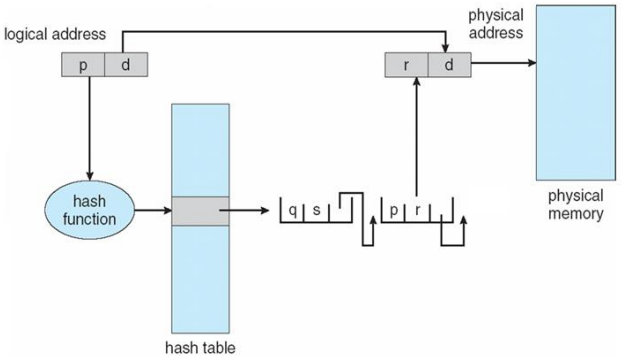
system needs 100 ns for main memory access while the cache access time is 10 ns. If the three-level hierarchical paging system is implemented with 80% hit ratio on average, answer its effective access time with calculation steps. (2pt)

-> For the cache hit, the memory access consists of m + e, 110 ns.

For the cache miss, the initial cache access, three page table accesses, and the main memory access are required as 4m + e, 410 ns.

Therefore, overall EAT becomes (m+e) * 0.8 + (4m+e) * 0.2 = 110ns * 0.8 + 410 * 0.2 = 88ns + 82ns = 170ns

c. Assume that you are going to implement a hashed page table system. To make it more efficient than the three-level hierarchical paging system in 2-b, you have designed a hash function that minimizes the hash collisions so your hash function makes only two collisions in each entry while the hash function calculation takes 5 ns. If the first chain hits 80% on average, what should be the minimum cache hit ratio to outperform the 3-level paging system in 2-b? Answer with EAT calculation. (3pts)



-> Similarly, the cache hit requires one memory access and one cache access, m + e.

For the cache miss, two scenarios exist:

1) 1st chain hit: the initial cache access occurs, hash function calculation, hash table access, first chain comparison, main memory access

e + h + 3m

2) 2nd chain hit: the initial cache access occurs, hash function calculation, hash table access, first chain comparison, second chain comparison, main memory access

e + h + 4m

EAT = (m + e)a + (e + h + 3m)(1-a)0.8 + (e + h + 4m)(1-a)0.2 <= 170

= (m – h – 3.2m)a + (e + h + 3.2m)

= (-2.2m –h)a + (e + h + 3.2m) <= 170

= -225a + 335 <= 170

= -225a <= -155

a >= 155/225 = 31/45 ~= 68.89%

3. Virtual Memory (12pts)

a. Explain what the copy-on-write is and describe the behaviors of vfork() 1) if the child process immediately executes another program and 2) if the parent process runs before the child process executes another program and write on its page. Consider the system uses pure demand paging. (3pts)

-> Copy-on-write is a technique that both the child and parent process initially share the same pages in the memory, so the creation of a child process is less expensive in comparison to the ordinary fork() which copies the entire pages of the parent to its child process.

1) If the child process immediately executes another program, then the child process builds its new page table differing to the parent's page table, which was the same before execution, and tries to upload the demand pages into main memory.

2) If the parent process runs prior to the child process' execution, then the parent process' modification on a page triggers copying the page to the other frame and the copied frame is modified. At the same time, the child process' page table direct to the unmodified page.

b. Let's assume that we have the memory reference string here:

0 1 2* 3 4* 2 3 4 0 3* 5 6* 0 3 4

where n^* represents the referencing modifies the page.

Assume that a system allocates three frames at the process creation and reallocates frames by its working set of previous five pages referenced. When the reallocated frame decreases, the victim frame is selected by the priority of 1) not filled with, 2) frame including a page outside the working set, and 3) least recently accessed. Assume that this system requires 1ms for handling the page fault, 10ms for reading a single page in, 20ms for writing a single page out, and ignorable duration (~= 0ms) for the other processing. Also, paging out for the unmodified pages simply zero-fill the page, not writing out to the backing store.

(i) How many page faults occur with FIFO and how long does it take to run the memory string? (3pts) **(Answering only the number is scored as 0.5pt.)**

-> 10 page faults, 55 + 11 + 104 = 170ms

Page	**0**	**1**	**2***	**3**	**4***	**2**	**3**	**4**	**0**	**3***	**5**	**6***	**0**	**3**	**4**
Fault	F	F	F	F	F				F		F	F		F	F
Time	11	11	11	11	11	-	-	-	11	-	31	31	-	31	11
FIFO 1	0	0	0	1	2*	2*	2*	2*	2*	2*	3*	4*	4*	0	5
FIFO 2		1	1	2*	3	3	3	3	3	3*	4*	0	0	5	6*
FIFO 3			2*	3	4*	4*	4*	4*	4*	4*	0	5	5	6*	3
FIFO 4									0	0	5	6*	6*	3	4
FIFO 5															

(ii) Assume that a system delays writing out a modified page for four memory references; the delayed dirty page returns to the frame if the process refers the page within four memory references, otherwise write out it to the backing store. When the dirty page comes back to the frame, no page-in cost occurs but only the page-fault cost exists. How long does it take with FIFO? (3pts)

-> 55 + 11 + 44 = 110 ms

At the 11th reference, 2* is not written out and put into the delayed queue. At the 12th reference, 3* is also not written out. At the 14th reference, 3* is re-referenced and 3* returns to the frame with the cost of PF (1ms). At the 15th reference, 2* is not referenced for four references so it should be written out (20ms), and 4* is in the dirty page queue, so no page-in cost occurs (1ms).

Page	0	1	2*	3	4*	2	3	4	0	3*	5	6*	0	3	4
Fault	F	F	F	F	F				F		F	F		F	F
Time	11	11	11	11	11	-	-	-	11	-	11	11	-	1	21
FIFO 1	0	0	0	1	2*	2*	2*	2*	2*	2*	3*	4*	4*	0	5
FIFO 2		1	1	2*	3	3	3	3	3	3*	4*	0	0	5	6*
FIFO 3			2*	3	4*	4*	4*	4*	4*	4*	0	5	5	6*	3*
FIFO 4									0	0	5	6*	6*	3*	4*
FIFO 5															

Dirty Page 1											2*	2*	2*	2*	
Dirty Page 2												3*	3*	4*	
Dirty Page 3															

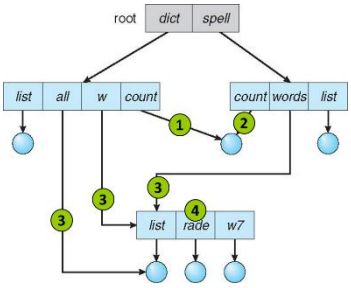
(iii) In the system same as 3-b(ii), how many PFs occur and how long does it take if Optimal Algorithm is applied? (3pts)

-> 8 page faults, 55 + 11 + 42 = 108 ms

Page	0	1	2*	3	4*	2	3	4	0	3*	5	6*	0	3	4
Fault	F	F	F	F	F				F		F	F			
Time	11	11	11	11	11	-	-	-	11	-	11	11	-	-	20
Frame 1	0	0	0	0	4*	4*	4*	4*	4*	4*	4*	4*	4*	4*	4*
Frame 2		1	1	3	3	3	3	3	3	3*	3*	3*	3*	3*	3*
Frame 3			2*	2*	2*	2*	2*	2*	2*	2*	5	6*	6*	6*	6*
Frame 4									0	0	0	0	0	0	0
Frame 5															

Dirty Page 1											2*	2*	2*	2*	
Dirty Page 2															
Dirty Page 3															

4. File-system (8pts)



a. In the right figure, let's assume that a user deleted (1) pointer. Then, answer the name of (2) and one solution not to make (2) in the file system. Also, if a user deleted (3) pointers, then explain what OS does for handling (4) and answer the procedure's name. (2pts)

-> (2) -> Dangling pointer

Solutions: by keeping backpointers using the daisy-chain structure, so if a user deletes a file then OS can delete all pointers for the file or delete the commanded link only.

By keeping a counter of pointers in a file or a directory, the deleting operation decreases the counter and the file is completely deleted if the counter becomes zero.

(4) OS traverses all files and directories first while marking them, and scan files and directories that cannot be accessed from its root directory. This procedure is called as garbage collection.

b. Let's say a user process runs two following codes:

```
myFile = fopen("c:\OS\finalExam.txt", "r");
fscanf(myFile, "%s\n", inString);
```

Please explain the procedure of access, open, and read the file using the in-memory file system structures. (4pt)

-> At the secondary drive mounting, the directory structure is uploaded in the kernel memory. At the first code, the function call of fopen() is delivered to the system call open() via dual-mode operation, and the kernel thread resolves the path to find where the file control block of "finalExam.txt" exists. After uploading the file control block to the kernel space from the disk through I/O, the kernel stores the pointer to the file control block in the system-wide open file table and updates the process-wide open-file table. After returning to the user mode, the process receives the index of the process's open-file table as a file handle.

Then, for fscanf, a user process sends the file handle with parameters. The relevant system call, read(), runs after changed to the kernel mode and refers the process-wide and system-wide open-file tables using the file handle to find the address of the file control block. With the parameters, the routine of read() now calculates the relative disk location of the target data and sends it to the disk I/O with the address of inString and change the process state as waiting. After the reading I/O, the process is woken up and return to the user mode, and the user process can access the data through inString variable.

c. In comparison to the contiguous allocation, explain the advantages of the extent-based systems. Also, describe what happens in the extent-based systems for the long-time usage. (2pts)

-> Extent-based systems divide the contiguous file into several sub-contiguous parts so mostly provide an easy way of being allocated in the secondary disk with no external fragmentation while keeping fast read and write speeds coming from the contiguous allocation. However, the

long-time usage leads to the increased number of extents which requires more data to store the metadata of extents and potentially slower read and write speeds to move the disk magnetic arm to the different extent locations.

Good Luck!