PDS/SDP OS internals 02/09/2024

Question 1.

1. In an inverted page table system, can a single page table entry be shared by multiple virtual pages from different processes without causing a conflict in address translation?? (YES/NO, Briefly explain/motivate)

   Yes, In the base version of an IPT, where an IPT entry just holds the page number and the PID, this would not be possible, as one given frame could belong to just one process. But this would clearly be a problem for shared pages/libraries. As IPTs are typically integrated with hash tables, most IPTs also hold one pointer, which is usually exploited for hash collision chains/lists. The standard hash-based solution would not be enough to share one IPT entry by different processes (due to how hash functions work, the entries would not be in the same collision list of the hash). So more complex solutions are needed from software, but this can be done: essentially the IPT pointer can be used, in case of shared frames, for a list of pages sharing the frame.

   In the exam, a certain flexibility has been allowed in the answer

2. In a system that uses local page replacement, can a process with a very high page fault rate still cause a reduction in the physical memory available to other processes?(YES/NO, Briefly explain/motivate)

   No, In local page replacement, each process is allocated a fixed set of physical frames, and its page faults only cause page replacements within its own allocated frames. This means that one process's high page fault rate does not directly reduce the physical memory available to other processes.

3. In a virtual memory system with a fixed degree of multiprogramming, can thrashing still occur if the sum of all processes' working sets is less than the available physical memory but the page replacement algorithm is not optimized? (YES/NO, Briefly explain/motivate)

   Yes, Thrashing can occur if the page replacement algorithm does not correctly predict or manage the working sets of the processes, leading to frequent unnecessary page replacements. This inefficiency can cause processes to repeatedly evict pages that will soon be needed again, resulting in excessive page faults and thrashing, despite sufficient physical memory.

4. Consider a 64-bit system with a virtual address space of 2^{48} bytes, a page size of 4 KB, and 16 GB of physical memory. Assume the system uses an inverted page table. Calculate and explain the following:

A. The number of bits required for the physical frame number.
B. The total number of entries in the inverted page table.
C. The virtual address given to a process is 0x00007FFFFFFFF000. Determine the physical address if this virtual address is mapped to the 1,024th physical frame. Show all steps, including the calculation of page number and offset.

Answers:
A. Physical Frame Number Calculation:

- Page size = 4 KB = 2^{12} bytes.
- Physical address space = 16 GB = 2^{34} bytes.
- Number of physical frames = $2^{34}/2^{12} = 2^{22}$.
- Therefore, the number of bits required for the physical frame number is 22 bits.

B. Total Number of Entries in the Inverted Page Table:

- Total number of entries equals the number of physical frames: 2^{22} entries.

D. Physical Address Calculation:

Step 1: Break down the virtual address into page number and offset:

- Virtual Address: 0x00007FFFFFFFF000
- Page size = 2^{12} bytes, so the offset within the age is lower 12 bits.
- Offset = '0x000' (last 12 bits)
- Virtual Page number: '0x00007FFFFFFFF', the remaining upper bits.

Step 2: The 1,024th physical frame corresponds to a physical frame number of 1024 (in binary: 0x00000400).

Step 3: Combine the physical frame number with the offset to get the physical address:

- Physical Address = (Physical Frame Number << 12) | Offset
- Physical Address = 0x00000400 << 12 = 0x00000400000
- Final Physical Address = 0x00000400000 + 0x000 = 0x00000400000

Question 2.

Consider a file system using **linked allocation** for file storage. The disk is divided into blocks of 4 KB each. Each block contains a pointer to the next block, which occupies 4 bytes of the block. A file requires 5 MB of storage.

A. Calculate the number of disk blocks required to store the entire file, including the space needed for the pointers.
B. If the pointers were reduced to 2 bytes (assuming smaller disk capacity), recalculate the number of blocks required. Would this change make the file storage more efficient in terms of block usage?
C. Compare the overhead introduced by the pointers in both cases as a percentage of the total storage used.

1:

- File size = 5 MB = 5×1024 × 1024 x 1024 bytes = 5,242,880 bytes.
- Block size = 4 KB = 4096 bytes.
- Each block has 4096 - 4 = 4092 bytes available for file data due to the 4-byte pointer.
- Number of blocks required = 5,242,880 bytes/ 4092 (bytes/block)
- Number of blocks = 1280

2:

- Now the pointer is 2 bytes, so each block has 4096 - 2 = 4094 bytes available for file data.
- Number of blocks required == 5,242,880 bytes/ 4092 (bytes/block) = 1281,25.
- Number of blocks = 1282.
- With pointers of size 2 one block is saved: 5,242,880 bytes/ 4094 (bytes/block) = 1280,63 -> 1280 blocks.

4-byte pointer overhead:

Total Overhead = 1282 x 4 bytes = 5128 bytes.

Percentage of overhead = 5128 bytes / 5,242,880 x 100 ≈ 0.098%.

2-byte pointer overhead:

Total overhead = 1281 x 2 bytes = 2562 bytes.

Percentage of overhead = 2562 bytes / 5,242,880 bytes x 100 = ≈0.049%.

3:

The overhead is halved when using a 2-byte pointer, reducing it from approximately 0.098% to 0.049% of the total storage.

Question 3.

A. In a system with multiple I/O devices and a single CPU, can I/O operations occur in parallel if the CPU is busy executing a process?

Yes,. I/O operations can be handled by dedicated I/O controllers, allowing them to proceed independently of the CPU, which can continue executing processes while I/O operations are ongoing.

B. In a system with demand-paging virtual memory and DMA, can a page fault occur during a DMA transfer, causing the transfer to fail if proper precautions are not taken?

Yes. The page fault is not triggered by the DMA transfer, as DMA only handles physical addresses (so contiguous physical memory). But a page fault can be triggered by code (of the same or other process) running in parallel. If this happens, pages involved in DMA should not be selected as victim for replacement. The operating system should thus either lock the pages in memory during the DMA transfer or handle page faults in such a way that does not disrupt the DMA process.

C. In a system using interrupt-driven I/O, is it possible for an interrupt to be missed if the interrupt controller is busy processing another interrupt and the device that triggered the second interrupt does not support interrupt queuing?

Yes. If the interrupt controller is busy processing another interrupt and the device that triggered the second interrupt does not support interrupt queuing, the second interrupt can be missed. This occurs because the interrupt controller may not be able to register the new interrupt while handling the current one, and without queuing, the device cannot hold the interrupt request until the controller is ready.

D. Consider a Hard Disk Drive (HDD) with the following specifications:

- Sector size: 512 Bytes
- Number of tracks per surface: 5,000
- Number of sectors per track: 300
- Number of double-sided platters: 6
- Platter rotation speed: 7,200 rpm (revolutions per minute)

1. Calculate the maximum possible data transfer rate in megabytes per second (MB/s), assuming one track of data can be transferred per revolution.

   Track Capacity=300×512 Bytes=153600 Bytes=0.1465 MB
   Data Transfer Rate = 0.1465MB/8.33 ms x 1000 = 17
   .57 MB/s
   (in this solution we are assuming that traces on a given cylinder are not read/written in parallel: this depends on the HDD low level technology used, so there could be an alternative solution differing by a factor 12: we prefer the proposed solution as the text clearly states “one track of data per revolution”)

2. If the disk experiences a head crash on one platter, how does this impact the total capacity and the data availability assuming no RAID or backup mechanisms are in place?

   Losing one platter reduces total capacity by 2/12 (since each platter has two surfaces) leading to a new capacity of 9.216 x 10/12 = 7.68 GB.
   Data on the crashed platter is lost and cannot be recovered without backups.

3. If the disk has an average seek time of 4 ms and needs to read a 1 GB file that is split across 200 non-contiguous tracks, calculate the total time required to read the file. Include the time for seeking, rotational latency, and data transfer. Assume that the average rotational latency is 4.165 ms and that one track can be read per revolution.
   The proposed exercise includes an error: 200 tracks (and cylinders) cannot hold a file of 1GB, as their maximum capacity is 200*12*300*512 Bytes = 200*300*6 Kbytes = 350 MB (approximated). So this has been considered in evaluating the exercise. Here we propose a solution that ignores this problem.

   Total Time to Read 1 GB File Across 200 Non-Contiguous Tracks:

   Step 1: Calculate the data transfer time:

   File Size=1 GB=1024×1024×1024 Bytes=1,073,741,824 Bytes.

   Total Data Transfer Time = 1,073,741,824 Bytes/ 153600 Bytes per Track X 8.33 ms per Track ≈58,214.63 ms=58.21 s (here the number of tracks considered is 1,073,741,824/(12*153600) = 583)

   In the rest of the exercise we use 200 tracks. The exercise could be easily modified to consider 583 tracks or more.

   Step 2: Calculate the total seek time:

   Total Seek Time=200×4 ms=800 ms=0.8 s

   Step 3: Calculate the total rotational latency:

   Total Rotational Latency=200×4.165 ms=833 ms=0.833 s

Step 4: Sum all times:

Total time = 58.21s + 0.8s + 0,833s = 59.8s

Question 4.

An oS161 system is given, running on a sys161 MIPS simulator with 8MB of RAM memory.

Physical memory in dumbvm is allocated by calling function ram_stealmem. For each of the following sentences, answer true/false (no explanation needed)

Function ram_stealmem incrementally allocates memory starting at the first available address after loading the kernel
TRUE
The allocation scheme used by ram_stealmem will not allow interleaving of kernel and user physical RAM contiguous partitions (in other words kernel partitions and user partitions will never be interleaved)
FALSE
Function ram_stealmem is called for memory allocation of both kernel (dynamic memory) and user processes
TRUE
Though the solution proposed in order to support memory de-allocation (freeppages) is based on a bitmap, a linked list could be used as well
TRUE
When using a bitmap in order to keep track of allocated/free RAM, a second array storing the sizes of allocated partitions is required for allocation of both kernel dynamic memory and user processes
FALSE
The stack of a user process is not allocated (in dumbvm) calling ram_steelmem, because the size of the stack is a constant, and the stack is always located at the same logical addresses.
FALSE

It is known that a user address space is characterized by as->as_pbase1, as->as_pbase2, as->as_vbase1, as->as_vbase2, as->as_npages1, as->as_npages2, as->as_stackpbase, having the following values: 0x200000, 0x300000, 0x3000, 0x7000, 3, 4, 0x400000. Pages in OS161 have size 4KB. It is also known that a user stack is allocated 18 pages in dumbvm.
Given the following logical addresses (they could be either user or kernel) convert them to the related physical addresses.
- 0x8110 -> (in segment 2) 0x8110-0x7000 + 0x300000 = 0x301110
- 0x6500 -> invalid because 0x6500-0x3000 > 3 pages
- 0x7FFFE010 -> (user stack) 0x7FFFE010 – (0x80000000-0x12000) + 0x400000 = 0x10010 + 0x400000 = 0x410010
- 0x805000B0 -> (kernel address) 0x805000B0 – 0x80000000 = 0x5000B0

Question 5.

A) Consider function sys_getpid (prototype below)

pid_t sys_getpid(void);

implemented in lab4. For each of the following sentences, answer true/false (no explanation needed)

1) The function is needed in order to implement sys_waitpid, because the pid of the waiting process is needed FALSE
2) The function is needed in order to implement the getpid system call, and it will be directly called by a user process FALSE
3) The function is needed in order to implement the getpid system call, and it will be called by the kernel, in function syscall TRUE
4) The function returns the pid of the current process TRUE
5) The function is called to obtain the pid of another process (the one to be waited for termination) FALSE
6) The function is called by sys__exit, in order to know the pid of the process to be signalled FALSE

B) A version of function proc_wait, provided within the proposed solution in lab 4, is provided. The function contains 3 errors. Find them and, for each of them, provide an explanation (why are they errors?)

```
int proc_wait(struct proc *proc)
{
```

```
    int return_status;
      KASSERT(proc != NULL);
      KASSERT(proc != kproc);

    cv_wait(proc->p_cv,proc->p_lock);
    lock_acquire(proc->p_lock);
    return_status = proc->p_status;
    if (return_status < 0)
             return return_status;
    lock_release(proc->p_lock);
    proc_destroy(curproc);
    return return_status;
}
```

	Error	Explain
1	Lock_acquire has to be called before cv_wait	The lock has to be owned when calling cv_wait on a condition variable
2	The conditioned return before lock_release is wrong	Not only the return is redundant and skipping proc_destroy, A return inside a critical section (lock_acquire … lock_release) is a potential source of deadlock (because the lock is not released upon return)
3	proc_destroy should be called with parameter proc (not curproc)	The waiting process should destroy another process, not itself (this would be a big problem!)