PDS OS internals 22/01/2024 – STANDARD version (with proposed solution)

PDS: L'esame è comune ai corsi PDS e SDP (in inglese). Si è predisposta un'unica proposta di soluzione in inglese.

1 In the context of virtual memory management, consider a system with a paging scheme:

A) Define what is a page fault, explain the conditions under which a page fault occurs and the consequences for the operating system.

B) List the steps involved in handling a page fault, elaborating on how the operating system responds to a page fault, and the mechanism it employs to resolve the faults.

C) Consider the two-dimensional array A: int A[100][100], where A[0][0] is at location 800 in a Byte addressable, paged memory system with pages of size 800 Bytes. The size of an integer is 32 bits. The process that manipulates the matrix resides on page 0 (locations 0 to 799). Thus, every instruction fetch will be from page 0. For three-page frames, how many page faults are generated by the following array initialization loop? Use LRU replacement and assume that one page frame contains the code and the other two are initially empty.

1.
```
For (int j=0; j <100, j++)
    for (int i = 0; i < 100; i ++){
        A[i][j] = 0;
        }
```

2.
```
For (int i=0; i <100, i++)
    for (int j = 0; j < 100; j ++){
        A[i][j] = 0;
        }
```

Answers

A. Define what is a page fault, and explain the conditions under which a page fault occurs and the consequences for the operating system.

A page fault occurs when a program attempts to access a page that is not currently in the main memory. This can happen due to various reasons, such as being swapped out to disk, or it has never been brought into memory before. When a page fault occurs, the operating system initiates a page fault handler to address the issue.

B. List the steps involved in handling a page fault, elaborating on how the operating system responds to a page fault, and the mechanism it employs to resolve the faults.

Handling a page fault involves several steps. When a page fault is detected, the operating system follows these steps:

1. Reference, 2. Trap, 3. Page is on backing store, 4. Bring in missing pages. 5. Update page table, 6. Resume process execution

C. Consider the two-dimensional array A: int A[100][100], where A[0][0] is at location 800 in a paged memory system with pages of size 800. The process that manipulates the matrix resides on page 0 (locations 0 to 799). Thus, every instruction fetch will be from page 0. For three-page frames, how many page faults are generated by the following array initialization loop? Use LRU replacement and assume that page frame 1 contains the process and the other two are initially empty.

1. Answer = 5,000. A page contains 2 rows of the matrix. As the outer iteration is on columns, the program visits all pages 100 times.
2. Answer = 50. The outer iteration is on rows, so once a page is replaced, it is never used again.

2

A) Explain the RAID structure, discuss its benefits, explaining how the chosen RAID level improves data storage and retrieval in comparison to a single disk.

B) In the context of RAID structure, define the Mean Time to Failure (MTTF), Mean Time to Data Loss (MTTDL), and Mean Time to Repair (MTTR). Provide an explanation of each term and elaborate on how these metrics contribute to the overall reliability and availability of a RAID system.

C) Consider a RAID structure with 4 disks in a mirrored (RAID 1) configuration. Assuming each disk can fail independently of the others, with an MTTF of 50000 hours and MTTR of 10 hours, calculate the MTTDL. Provide a step-by-step explanation of the calculation process.

Answers

A) Explain the RAID structure, discuss its benefits, and explain how the chosen RAID level improves data storage and retrieval compared to a single disk. Explain in detail, RAID 1, mirroring .

RAID, or redundant Array of Independent Disks, is a storage technology that combines multiple disk drive components into a logical unit to improve data performance, reliability, and/or fault tolerance.

B) In the context of RAID structure, define the Mean Time to Failure (MTTF), Mean Time to Data Loss (MTTDL), and Mean Time to Repair (MTTR). Provide an explanation of each term and elaborate on how these metrics contribute to the overall reliability and availability of a RAID system.

Mean time to Failure: represents the average time between the start of an operation of a system and its failure, For RAID, it reflects the average time it takes for a disk to fail.
Mean Time to Data Loss: is the average time it takes for the system to experience data loss due to a disk failure.
Mean Time to Repair: is the average time required to repair a failed component and restore the system to its normal operational state. In RAID, it measures the time it takes to replace a failed disk and rebuild the array.

C) Consider a RAID structure with 4 disks in a mirrored (RAID 1) configuration. Assuming each disk can fail independently of the others, with an MTTF of 50000 hours and MTTR of 10 hours, calculate the MTTDL. Provide a step-by-step explanation of the calculation process.

$MTTDL = MTTF^2/(4*MTTR) = 50000^2/(4*10)\ h = 25/4*10^7\ h = 6.25*10^7\ h$

Note: We have 2 pairs of mirrored disks.
Whether you have 2 disks or 4 disk in a RAID 1 mirrored configuration, the MTTDL is associated with the scenario where both disks in any one of the mirrored pairs fail.
The meant time to a failure on any of the 4 independent disks is MTTF/4 (not MTTF/2).

3

Consider the usage of linked lists and bitmaps for free space management, either for free frames (in memory management) or free blocks (in file systems). Answer yes/no (true/false) to the following statements and provide the motivation for the answer.

A) Bitmaps are better than linked lists when searching for intervals of contiguous frames/blocks.
B) In a system with paging, free lists are the best solution for both process and kernel memory.
C) Free blocks for a file system based on i-nodes are better handled with bitmaps.
D) Searching for a free block in a (linked) free list has a linear (in the list size) cost with memory management based on a two-level page table.
E) A sorted linked free list has a linear (in the list size) cost both for contiguous space allocation and free.
F) A bitmap, though requiring extra memory, provides better time performance (than a free list) with a file system based on FAT (File Allocation Table)

Answers

A) Bitmaps are better than linked lists when searching for intervals of contiguous frames/blocks.

Yes. Because lists are linear, while bitmaps can have logarithmic performance, with proper search algorithms

B) In a system with paging, free lists are the best solution for both process and kernel memory.

NO. They are the best solution for process, but not for kernel, which needs contiguous memory (for page table allocation), whereas processes do not need contiguous pages (and linked lista are O(1)).

C) Free blocks for a file system based on i-nodes are better handled with bitmaps.

NO. Because no contiguous memory is needed, so lists are O(1), while bitmaps can have logarithmic performance at best.

D) Searching for a free block in a (linked) free list has a linear (in the list size) cost with memory management based on a two-level page table

NO. Page Tables do not need contiguous memory, so lists are O(1).

E) A sorted linked free list has a linear (in the list size) cost both for contiguous space allocation and free.

Yes. Because freeing memory implies an insert operation in a sorted list, and an allocation requires a search operation in the list: both operations are linear.

F) A bitmap, though requiring extra memory, provides better time performance (than a free list) with a file system based on FAT (File Allocation Table)

NO. Because FAT does not need contiguous allocation so a free list (directly implemented in the FAT) is O(1).

4

Consider a possible implementation of the *read()* system call in OS161.

A) Can the read destination be in kernel memory, or does it have to be (always) in user memory?

B) Suppose two user processes, running concurrently, open the same file for reading and call read(fd,v,N), on that file, do you expect that (for each option answer YES/NO and motivate):

1) fd is the same (integer number) for both processes.
2) The two processes read the file independently from each other (or they share the same read flow/sequence?).
3) A lock is needed, because operations have to be done in mutual exclusion.
4) If the implementation of the read system call uses a kernel buffer, the two processes will use the same buffer.

C) Given the following implementation of sys_read, for read operations on file

```
static int
file_read(int fd, userptr_t buf_ptr, size_t size) {
  ...
  kbuf = kmalloc(size);
  uio_kinit(&iov, &ku, kbuf, size, of->offset, UIO_READ);
  result = VOP_READ(vn, &ku);
  if (result) {
    return result;
  }
  of->offset = ku.uio_offset;
  nread = size - ku.uio_resid;
  copyout(kbuf,buf_ptr,nread);
  kfree(kbuf);
  return (nread);
}
```

Answer the following questions:

1) Does the operation exploit a kernel buffer? (motivate)
2) Is it possible that the return value differs from the value of size parameter? (motivate)
3) Could kmalloc be called with a parameter < size (e.g. kmalloc (size/2))? (motivate)

Answers

A)

It has to be in user memory, as this is a system call for user processes.

B)

1) NO. Processes ca have different fd numbers. The file descriptor depends on the sequence of open done by a process.
2) YES, Reads are independent, because each process did a file open, so it has its own openfile struct.
3) NO. Because they just do read operations.
4) It depends on the implementation. If the kernel buffer is dynamically allocated (and freed) within the system call, buffers are not shared. The buffer could be shared if implemented as a global/unique buffer, but this solution would need a proper synchronization (mutual exclusion).

C)

1) YES. Kbuf is points to a kernel buffer.
2) YES. Size is the maximum number of Bytes read, but the file could end before reaching this number.
3) YES. But this would require multiple VOP_READ operations in order to read “size” Bytes. The proposed implementation reads all Bytes in a single VOP_READ, so if we keep a single VOP_READ the answer is NO..

5 An OS161 system is given.

A) Consider the functions thread_switch and thread_yield. Are they equivalent, or do they perform different tasks? (motivate)

B) when a new kernel thread is created in OS161, does it run immediately or is enqueued for running?

C) The implementation of thread_exit() ends with a call

```
switchframe_switch(&cur->t_context, &next->t_context);
```

What do the two parameters represent (it is not enough to say “thread context”)? Which of the two parameters refers to the saved context and which one to the restored context?

Answers

A)

Though similar, they perform different tasks. Both functions will end up scheduling another thread for execution, but thread_switch is more general. Thread_yield can be viewed as a particular case of thread_switch, where the new state of the current thread is READY.

B)

It is enqueued, as t will not be able to un until the next thread_switch.

C)

The two parameters are pointers to the thread context saved in the thread kernel stack. The context (registers) of the current thread is SAVED in the switchframe pointed by the fist parameter, the context of the new thred is restored (READ) from the switchframe pointed by the second parameter.