System and device programming

OS internals Exam - 19/6/2021 (Cabodi) - standard version

Iniziato	sabato, 19 giugno 2021, 08:27
Terminato	sabato, 19 giugno 2021, 10:06
Tempo impiegato	1 ora 38 min.
Valutazione	**11,50** su un massimo di 15,00 (**77**%)

Domanda 1

Completo

Punteggio ottenuto 3,00 su 3,00

WHEN RESULTS ARE NUMBERS, RELEVANT INTERMEDIATE STEPS (OR FORMULAS) ARE NEEDED

ALL YES/NO ANSWERS MUST BE EXPLAINED/MOTIVATED

Consider the following string of memory references, for a given process. For each reference (Byte addressing, with addresses expressed in hexadecimal code) the read(R)/write(W) operation is also reported: R 63F5, R 5A64, W 1AD3, W 6E7E, R 18C8, W 23D1, R 194E, R 5465, W 62A0, R 3BBA, W 1CE6, R 1480, R 6294, R 5AB8. Assume that physical and logical addresses are on 16 bits, page size is 4KBytes, and A817 (Hex) is the maximum address usable by the program (the address space top limit).

A) Compute the size of the address space (expressed as number of pages), and the internal fragmentation.

B) Simulate an LRU ((Least Recently Used) page replacement algorithm, with a limit of 3 available frames. Represent the resident set (physical frames containing logical pages) after each memory reference.

C) Show page faults (references to pages outside the resident set) and compute their overall count.

D) Briefly explain why an exact implementation of the LRU algorithm would be inefficient, thus motivating an approximate version.

A) Compute the size of the address space (expressed as number of pages), and the internal fragmentation.

A817 is maximum address

page = 4KB = 0x1000

address space = ceil(0xA817 / 0x1000) = B = (dec) 11 pages

last page is used up to index 817. Internal fragmentation = 1000 - 817 = 7E9 = 2025 Bytes

B) Simulate an LRU ((Least Recently Used) page replacement algorithm, with a limit of 3 available frames. Represent the resident set (physical frames containing logical pages) after each memory reference.

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Accesses	6	5	1	6	1	2	1	5	6	3	1	1	6	5
Resident Set	6	6	6	6	6	6	6	5	5	5	1	1	1	1
		5	5	5	5	2	2	2	6	6	6	6	6	6
			1	1	1	1	1	1	1	3	3	3	3	5
Page Faults	*	*	*			*		*	*	*	*			*

C) Show page faults (references to pages outside the resident set) and compute their overall count.

Page faults are illustrated in the previous question

Total number of PF: 9

D) Briefly explain why an exact implementation of the LRU algorithm would be inefficient, thus motivating an approximate version.

The LRU alg can be implemented either by keeping a list of last recently used pages or by keeping track of the time of the last recent use for each cell of the resident set. The list has to be continuosly updated at each reference and this is an expansive operation. The timer implementation has to disadvantage that one should have to scan the whole resident set to find the one cell with the earlier clock time, thus the least reference; also this operation is time and resource consuming.

An approximate version could work by keeping a reference bit signaling whether the bit was referenced from the last page fault or not. The algorithm now scans the page and stops at the first reference bit it finds with the "non-referenced" value (0 or 1 depending on what the implementor decides). The solution is called approximate because it selects one of the Last Recently Used pages and not The Last Recently Used page.

A) Compute the size of the address space (expressed as number of pages), and the internal fragmentation.

The page size is 4KB, so 12 bits are needed for the offset/displacement

Total number of pages in the address space (including those ones not in the reference string):

A817 = 1010 1000 0001 0111.

The maximum page index is 1010 (binary) = 10(dec.) = A(hex) => So the address space has 11 pages.

Internal fragmentation: the last page is used up to the Byte at offset 1000 0001 0111 = 2K+23 = 2071

Overall, 2K+24 Bytes are used in the last page

Int,Fr. = 4K-(2K+24) = 2024 Bytes.

B) Simulate an LRU ((Least Recently Used) page replacement algorithm, with a limit of 3 available frames. Represent the resident set (physical frames containing logical pages) after each memory reference.

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Accesses	6	5	1	6	1	2	1	5	6	3	1	1	6	5

Resident Set	6	6	6	6	6	6	**6**	5	5	**5**	1	1	1	1
		5	5	5	**5**	2	2	**2**	6	6	6	6	6	6
			1	1	1	1	1	1	**1**	3	3	3	**3**	5
Page Faults	*	*	*			*		*	*	*	*			*

C) Show page faults (references to pages outside the resident set) and compute their overall count.

Page faults are shown in the table.

Total number of PF: 9

D) Briefly explain why an exact implementation of the LRU algorithm would be inefficient, thus motivating an approximate version.

Because in order to determine the victim in the case of a PF it is necessary to look for the page with the oldest access time. This can only be achieved with techniques involving an operation at each memory access (therefore not only in the case of PF):

a) keep track, for each page, of the last access time (this is the operation to be repeated at each access) and then do a minimum search when looking for the victim

or

b) manage a stack of double linked pages, moving the accessed page at the top of the stack, at each access (this avoids the search for the minimum, but requires an expensive operation at each access)

Commento:
OK

Domanda 2

Completo

Punteggio ottenuto 2,75 su 3,00

ALL YES/NO ANSWERS MUST BE EXPLAINED/MOTIVATED. WHEN RESULTS ARE NUMBERS

THE FINAL RESULT, AND RELEVANT INTERMEDIATE STEPS (OR FORMULAS) ARE NEEDED

A given disk is organised with physical and logical blocks of given (same) size 4KB. The disk contains multiple partitions: partition A has size NB blocks, ant is formatted for a file system that statically allocates NM blocks for metadata (directories, file control blocks and a bitmap, for free space management), and ND blocks for file data.
Every bit in the bitmap corresponds to one of the ND data blocks. NM/4 metadata blocks are used for the bitmap.

Answer the following questions:

A) Compute the ratio ND/NM.

B) We know that the bitmap shows a ratio free/used blocks of 50% (so 1 free block for 2 used/allocated). Compute (as a function of NM) the maximum size of a contiguous interval of free blocks, when assuming the most favourable bitmap configuration.

C) Briefly explain why, depending on the disk technology (SSD vs. HDD), different disk scheduling policies could be needed for the disk (you don't need to explain/describe the scheduling strategies, just motivate them)

A) Compute the ratio ND/NM.

NM / 4 = bitmap blocks
block size = 4KB
bitmap bits = ND blocks = NM/4 * 4K * 8 = NM * 8K

ND/NM = NM * 8K / NM = 8K = 8192

B) We know that the bitmap shows a ratio free/used blocks of 50% (so 1 free block for 2 used/allocated). Compute (as a function of NM) the maximum size of a contiguous interval of free blocks, when assuming the most favourable bitmap configuration.

In the most favorable conf we have all used blocks close each other and all free blocks close to each other. No used block is in the middle of the free blocks interval.

N_{good} = NM *8K / 3 = NM * 2730

C) Briefly explain why, depending on the disk technology (SSD vs. HDD), different disk scheduling policies could be needed for the disk (you don't need to explain/describe the scheduling strategies, just motivate them)

The HDD tecnology includes a mechanical head that has to physically move from different areas of the tape to actually read data. Since the head moving requires more time for phisically distant areas inside the HHD and less time for closer areas having a policy that allow to run less distance can save a lot of time. For example instead of moving to all memory references in the order they come from memory a smart policy could be, given a set of memory references, accessing references from leftmost part of the tape to rightmost part of the tape.
SSDs on the contrary don't need such policies since they don't have a phisicaly moving head, thus don't need to optimize the order in which memory references are accesed.

A) Compute the ratio ND/NM.

|bitmap| = NM/4 blocks = NM/4*4K*8 bits = 8K*NM bits
every bit in the bitmap corresponds to one of the ND data blocks
ND = 8K*NM
ND/NM = 8K

B) We know that the bitmap shows a ratio free/used blocks of 50% (so 1 free block for 2 used/allocated). Compute (as a function of NM) the maximum size of a contiguous interval of free blocks, when assuming the most favourable bitmap configuration.

N_{free}/N_{used} = 0,5 => N_{free} = 1/3*(N_{free}+N_{used}) = 1/3*(N_{bits}) = 0.33*8K*NM bits = 2.67K*NM bits

The most favourable configuration is the one with all free blocks contiguous:
N_{good} = 2.67K*NM

C) Briefly explain why, depending on the disk technology (SSD vs. HDD), different disk scheduling policies could be needed for the disk (you don't need to explain/describe the scheduling strategies, just motivate them)

Because with HDD technology, based on magnetic disks with mechanical movements, it is advisable to serve access requests in order to minimize seek time and rotational latency: this generally leads to trying to reduce the difference between block (or cylinder) indexes of successive requests. On the contrary, SSD technologies do not have seek times and rotational latency, so you can avoid scheduling (NOOP) or limit yourself to taking into account the asymmetry between reads and writes, and/or the possible optimization for accesses to adjacent blocks.

Commento:
A/B) ok -> 2

C) you should at least mention the electronic technology for SSDs, and motivate the exclusion of alternative scheduling needs -> 0.75

Domanda 3

Completo

Punteggio ottenuto 1,25 su 3,00

WHEN RESULTS ARE NUMBERS, RELEVANT INTERMEDIATE STEPS (OR FORMULAS) ARE NEEDED
ALL YES/NO ANSWERS MUST BE EXPLAINED/MOTIVATED

A) Briefly explain the following file structures, within the framework of a given file system :

1) Sequence of bytes
2) Simple record structure
3) Complex structures, such as (for instance) index and relative files.

B) Of all the above file structures, show whether they support sequential and/or direct (random) access.

C) Which of the following file allocation/implementation strategies can be (or cannot be) used with the above listed structures (1, 2, and 3): contiguous allocation, linked list of blocks, FAT and indexed allocation (e.g. inodes) (motivate the answer).

A) Briefly explain the following file structures, within the framework of a given file system :

1) Sequence of bytes
2) Simple record structure
3) Complex structures, such as (for instance) index and relative files.

1) file is just a sequence of bytes

2) file is divided in records and can be scrolled record by record instead of byte by byte

3) there is an index stored in the file header that allows to identify and access certain parts of the file

B) Of all the above file structures, show whether they support sequential and/or direct (random) access.

1) Sequence of bytes: only sequencial access, there's no way to identify a certain point to access within the file

2) Simple record structure: sequential access

3) Complex structures: only direct access through the index

C) Which of the following file allocation/implementation strategies can be (or cannot be) used with the above listed structures (1, 2, and 3): contiguous allocation, linked list of blocks, FAT and indexed allocation (e.g. inodes) (motivate the answer).

those which can be used are:

1) linked list of blocks, FAT. File is stored in a sequence of blocks and each block (or the File Allocation Table in the case of the FAT) has a link to the next one, so the only way to access it is sequencially

2) linked list of blocks, FAT. File can both be stored in a sequence of blocks linked one another

3) indexed allocation. file is stored in different blocks and the FCB holds the indexes (and locations) to each of the block componing the file. Either in a direct or in an indirect way (e.g. FCB stores index of a block storing the indexes of the blocks)

WHEN RESULTS ARE NUMBERS, RELEVANT INTERMEDIATE STEPS (OR FORMULAS) ARE NEEDED - ALL YES/NO ANSWERS MUST BE EXPLAINED/MOTIVATED

A) Briefly explain the following file structures, within the framework of a given file system :

1) Sequence of bytes
2) Simple record structure
3) Complex structures, such as (for instance) index and relative files.

These are file formats seen and used at the application level.
1) All files can be viewed as a sequence of bytes
2) The record is a higher level abstraction than the byte. You can have fixed or variable length records: for example, text files with lines terminated by end-of-line (\n) are of the second type. A file will appear as a sequence of records,
3) More complex file structures include data and metadata, organized for example in headers and/or sections. Index files are an example: an index file (or the initial part of the file) contains a table with pairs (key, pointer) that allow efficient search: the pointer points directly to the data in the data (relative) file or in the second part of the file.

B) Of all the above file structures, show whether they support sequential and/or direct (random) access.

1. The byte-organized file allows both sequential and direct/random access (if you know which byte to access)
2. Files with fixed-length records allow both sequential and direct access (given the record number and size, the byte to be accessed is calculated). Variable-length records allow sequential access only
3. Files with a complex structure (e.g. indexes) allow both sequential and direct access, provided that (for direct access) it is possible to efficiently access the index and/or the header in which to find the pointer to the data)

C) Which of the following file allocation/implementation strategies can be (or cannot be) used with the above listed structures (1, 2, and 3): contiguous allocation, linked list of blocks, FAT and indexed allocation (e.g. inodes) (motivate the answer).

All three file types are compatible with all allocation strategies. These are formats managed at different levels: the first at the application level, while the last at the "logical file system" and "file organization module" level.

Commento:
A) missing the fixed/variable length record in 2 -> 0.75

B) 1 and 2 are wrong -> 0,5

C) Fully missing the issue of the 2 SW layers -> 0

Domanda 4

Completo

Punteggio ottenuto 1,50 su 3,00

ALL YES/NO ANSWERS MUST BE EXPLAINED/MOTIVATED. WHEN RESULTS ARE NUMBERS, THE FINAL RESULT, AND RELEVANT INTERMEDIATE STEPS (OR FORMULAS) ARE NEEDED

Consider a user process on OS161

A) Briefly explain the difference between the switchframe and the trapframe. Which one is used to start a user process? Which one to handle a system call? Which one for thread_fork?

B) Is the switchframe allocated in the user stack or the lernel-level process stack? (motivate)

C) Why are IO devices mapped to kseg1? Would it be possible to map an IO device in kseg0?

D) Can a user process perform a write(fd,buf,nb), where buf is an address in kseg0? What should the SYS_write syscall do in order to correclty handle this case?

A) Briefly explain the difference between the switchframe and the trapframe. Which one is used to start a user process? Which one to handle a system call? Which one for thread_fork?

the switchframe is used for holding the thread data during a context switch from that thread to another. The context of the thread which is not running anymore is stored in the switchframe and can be restored from there once the thread becomes running again.
The trapframe is used for when interrupts (such as I/O interrupts) arise. In that case the context of the running thread is stored in the switchframe and is restored after the interrupt service routing has completed its operations.
start user process -> trapframe
handle system call -> trapframe
thread_fork -> switchframe

B) Is the switchframe allocated in the user stack or the lernel-level process stack? (motivate)

It is allocated in the kernel.level process stack because it's not supposed to be modified/accessed by user processes directly.
The kernel provides them of the content of the switchframe when they have to context switch from a thread to another.

C) Why are IO devices mapped to kseg1? Would it be possible to map an IO device in kseg0?

In wouldn't be possible to map an IO device in kseg0 because of the different way the two memory areas are structured

D) Can a user process perform a write(fd,buf,nb), where buf is an address in kseg0? What should the SYS_write syscall do in order to correclty handle this case?

It can, since kseg0 is the user-reserved part of the memory in OS161. whereas if buf was in kseg1 this would be an illegal instruction since a user process cannot access the kernel side of the memory.
To handle the kseg0 case correctly the sys_write syscall has to do nothing special.

A) Briefly explain the difference between the switchframe and the trapframe. Which one is used to start a user process? Which one to handle a system call? Which one for thread_fork?

Both structures are used to save the process context (registers + other information): the switchframe is used for the context switch (a change of the process running on a cpu), the trapframe is related to a trap, which means that we are still in the context of the process but in kernel mode.
The runprogram function uses a trapframe to start a user process with mips_usermode.
A system call takes advantage of the trapframe, as it is triggered as a trap.
The thread_fork uses a switchframe as it prepares a thread ready to be put into the ready queue.

B) Is the switchframe allocated in the user stack or the lernel-level process stack? (motivate)

In the kernel stack, because it cannot be visible in user mode

C) Why are IO devices mapped to kseg1? Would it be possible to map an IO device in kseg0?

IO devices must be mapped in kernel space. They are mapped to kseg1 as it is not cached: an IO device cannot be read / written using the cache, as any read / write operation must be performed on the IO device. For the same reason, the device cannot be mapped to kseg0, which is cached.

D) Can a user process perform a write(fd,buf,nb), where buf is an address in kseg0? What should the SYS_write syscall do in order to correclty handle this case?

The user process must not be able to read a logical kernel-type address.
The buf pointer is the source of the write operation, while the destination is the file, which could be a normal file or the console.
In general, any legal memory address is correct as a source, but, given a user process, a legal pointer should be mapped to the user space, so kseg0 is forbidden. To handle this event correctly, the SYS_write syscall should check that the buf parameter is a user address, returning an error if not.

Commento:
A/B) ok -> 1,5
C/D) no. wrong answers -> 0

Domanda 5

Completo

Punteggio ottenuto 3,00 su 3,00

ALL YES/NO ANSWERS MUST BE EXPLAINED/MOTIVATED. WHEN RESULTS ARE NUMBERS, THE FINAL RESULT, AND RELEVANT INTERMEDIATE STEPS (OR FORMULAS) ARE NEEDED

Consider an OS161 kernel:

A) Briefly describe the main features of semaphores and locks, and the main differences between them.

B) What are wait_channels? Why are they used inside the kernel?

C) Function wchan_wakeone is listed here:

```
wchan_wakeone(struct wchan *wc, struct spinlock *lk) {
  struct thread *target;
  KASSERT(spinlock_do_i_hold(lk));
  target = threadlist_remhead(&wc->wc_threads);
  if (target == NULL)  { return; }
  thread_make_runnable(target, false);
}
```

Motivate the statement KASSERT(spinlock_do_i_hold(lk));
Why is a thread removed from the wc->wc_threads list?

A) Briefly describe the main features of semaphores and locks, and the main differences between them.

semaphores are supposed to be used for thread syncronization whereas locks are supposed to be used just for protecting critical sections. A lock can be used in combination with a condition variable and the condition variable can then be used for syncronization (cv_wait, cv_notifyOne and cv_notifyAll have to be called from inside the critical section protected by the lock).
Also semaphores have a counter inside them and so it is possible to wake up more than one waiting thread at a time (for example if you initialize the semaphore to a value greater than one). That way one could have a specific number of threads (e.g. 3) and no more than that running on a section at the same time, provided that the section is not critical and that critical subsections inside of it are protected by, for example, another semaphore.
Locks on the other hand can only be locked and unlocked, they don't have a counter.

B) What are wait_channels? Why are they used inside the kernel?

wait channels are used as a syncronization primitive inside the kernel. They are condition variables, so they must be used within a section protected by the lock and they must have the lock in order to operate correctly.

specifically the function wait_channel must release the lock to allow other threads to access the critical section and execute a wake call (e.g wake_one), whereas wake calls don't necessarely need the lock but they will verify (via KASSERT) that the lock is held by the caller since only those threads having the lock could call these functions.

C) Function wchan_wakeone is listed here:

```
wchan_wakeone(struct wchan *wc, struct spinlock *lk) {
  struct thread *target;
  KASSERT(spinlock_do_i_hold(lk));
  target = threadlist_remhead(&wc->wc_threads);
  if (target == NULL)  { return; }
  thread_make_runnable(target, false);
}
```

Motivate the statement KASSERT(spinlock_do_i_hold(lk));
Why is a thread removed from the wc->wc_threads list?

The KASSERT statement is needed to verify that the lock is held by the calling thread, because only the thread holding the lock should be able to call the wake_one function.
A thread is removed from the waiting list since the wake_one was called and that thread needs to be woken up (is not waiting anymore) by thread_make_runnable

A) Briefly describe the main features of semaphores and locks, and the main differences between them.

Locks and semaphores are synchronization mechanisms.
A semaphore has a more general use: it can be used to manage mutual exclusion from a critical section but not only, while the lock is only used to manage mutual exclusion from a critical region.
A semaphore is characterized by an integer that could handle multiple accesses to a critical section; A lock can be seen as a mutex, or special case of a semaphore in which the maximum number of resources is equal to 1 (binary semaphore).

B) What are wait_channels? Why are they used inside the kernel?

The Wait channel: is a kernel synchronization structure/primitive, which allows wait and signal operations (called wchan_sleep, wchan_broadcast, wchan_wakeone), connected to an observed condition, similar to the "condition variable".
It was introduced in OS161 as a low-level primitive associated with a spinlock (while the condition variable is associated with a lock).

The wchan can only be used by the kernel thread that has acquired the spinlock.

C) Function wchan_wakeone is listed here:

```
wchan_wakeone(struct wchan *wc, struct spinlock *lk) {
  struct thread *target;
  KASSERT(spinlock_do_i_hold(lk));
  target = threadlist_remhead(&wc->wc_threads);
  if (target == NULL)  { return; }
  thread_make_runnable(target, false);
}
```

Motivate the statement KASSERT(spinlock_do_i_hold(lk));
Why is a thread removed from the wc->wc_threads list?

The KASSERT (...) instruction is not strictly necessary for the execution of the program but is used to identify programmer errors because when the function in question is invoked, the spinlock must be absolutely owned.
For the wchan_wakeone behavior only one thread needs to be awakened. The list serves as more threads can be on hold. In this specific case, the first on the list is removed.

Commento:
ok

Domanda 6

Completo

Non valutata

Select "Withdraw from exam" if you wish to withdraw (your test won't be evaluated).

Don't answer or select "I want my exam evaluated" if you are completing your test for evaluation.

You'll be able to possibly withdraw later on, after the test is closed, once you see the proposed solution.

○ (a) Withdraw from exam (my test won't be evaluated)

[✓ ● (b) I want my exam evaluated]

Risposta errata.

La risposta corretta è: Withdraw from exam (my test won't be evaluated)