System and device programming

Iniziato	giovedì, 10 giugno 2021, 15:25
Stato	Completato
Terminato	giovedì, 10 giugno 2021, 15:25
Tempo impiegato	5 secondi
Valutazione	**0,00** su un massimo di 15,00 (**0**%)

Domanda 1

Risposta non data

Punteggio max.: 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 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 33F5, R 2A64, W 0AD3, W 2E7E, R 08C8, W 13D1, R 094E, R 3465, W 32A0, R 1BBA, W 0CE6, R 1480, R 3294, R 2AB8. Assume that physical and logical addresses are on 16 bits, page size is 2KBytes, and 3C02 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) Compute the string of page references.

C) Simulate a PFF (Page fault frequency) page replacement algorithm, assuming a single time threshold C=2. Represent the resident set (physical frames containing logical pages) after each memory reference. Explicitly indicate the activations of the algorithm (the victim selection procedure). Also represent (by a subscript or other notation) reference bits associated to pages/frames.

The (approximate) PFF algorithm can be summarized as follows:

- *The algorithm is activated at every page fault, not at page reference.*
- *Based on time interval TAU from previous page fault*
  - *if TAU < C, i.e. page fault frequency greater than desired, add a new frame to the Resident Set of the page faulting process.*
  - *if TAU >= C, i.e. page fault frequency OK, remove from Resident Set of page faulting process all pages with reference bit at 0; then clear (set 0) reference bit of all other pages in Resident Set.*

NOTES: when evaluating TAU, let's just consider the times occurring "between" the two page faults, so for instance when two page faults occur at times 7 and 8, TAU = 0, when two page faults occur at times 12 and 15, TAU = 2). Reference bits are set at each reference, and reset as the last step of the replacement algorithm. So, whenever reference bits are cleared, this includes the faulting page (the current reference)

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

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

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

3c02 = 0011 1,100 0000 0010.

The maximum page index is 00111 (binary) = 7(dec) => So the address space has 8 pages.

Internal fragmentation: The last page is used up to the Byte at offset 100 0000 0010 = 1K+2 = 1026

Overall, 1K+3 Bytes used in the last page

Int.fr. = 2K-(1K+3) = 1021 Bytes.

B) Compute the string of page references.

Logical addresses (p,d) (page,displacement).

A page contains 2KBytes, so the displacement (d) needs 11 bits, whereas p needs 5 bits.

References expressed in binary: R (0011 0,011 1111 0101), R (0010 1,010 0110 0100), W (000 1,010 1101 0011), …..

Reference string (p is enough, d not needed):

6 (2*3+0), 5 (2*2+1), 1, 5, 1, 2, 1, 6, 6, 3, 1, 2, 6, 5

notice that p, given by the 5 most significant bits of the logical address, can be quickly computed as the double of the first hexadecimal digit + the MSB (most significant bit) of the second hexadecimal digit).

C) Simulate a PFF (Page fault frequency) page replacement algorithm, assuming a single time threshold C=2. Represent the resident set (physical frames containing logical pages) after each memory reference. Explicitly indicate the activations of the algorithm (the victim selection procedure). Also represent (by a subscript or other notation) reference bits associated to pages/frames.

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13
References	6	5	1	5	1	2	1	6	6	3	1	2	6	5
Resident Set	6	6	6	6	6	6_0	6_0	6	6	6_0	6_0	6_0	6	6
		5	5	5	5	5_0	5_0	5_0	$\mathbf{5_0}$	3_0	3_0	3_0	3_0	3_0
			1	1	1	1_0	1	1	1	1_0	1	1	1	1
						2_0	2_0	2_0	2_0			2	2	2
														5
Page Faults	*	*	*			*				*		*		*
Algorithm activation						x				x				

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

Page faults have been shown in the table.

Total amount of PF: 7

Domanda 2

Risposta non data

Punteggio max.: 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 Unix-like file system, based on inodes, with 13 pointers/indexes (10 direct, 1 single indirect, 1 double indirect and 1 triple indirect). Pointers/indexes have size 32 bits, and disk blocks have size 2KB. The file system resides on a disk partition of size 400GB, ***that includes both data blocks and index blocks***.

A) Assuming that all metadata (except index blocks) have negligible size, compute the maximum number of files that the file system can host, using single indirect indexing (N1), and using triple indirect indexing (N3).

B) Given a binary file of size 20490.5KB, compute exactly how many index blocks and data blocks the file is using.

C) Consider the same file of Q B, where the lseek(fd,offset,SEEK_SET) operation is called to position the file offset for the subsequent read/write operation. Suppose fd is the file descriptor associated to the file (already open), that offset = 0x00800010. Compute the logical block number (relative to the file blocks, numbered starting at 0) at which the position is moved. If the file uses a single (or double/triple) indirect indexing, compute which row of the outer index block contains the data block index (or the inner index block index).

A) Assuming that all metadata (except index blocks) have negligible size, compute the maximum number of files that the file system can host, using single indirect indexing (N1), and using triple indirect indexing (N3).

General observations
An index block contains 2KB/4B = 512 pointers/indexes.
The partition contains 400GB/2KB = 200 M blocks

Computing maximum numbers N1/N3 *(for files with* single/triple indirect indexing)
In order to compute the maximum number of files, we need to consider the minimum occupancy. We need to consider both data blocks and index blocks.

Let's use MIN_1 and MIN_3 for minimum occupation of the two kinds of file:

MIN_1 = (10 + 1) data blocks + 1 index block
MIN_3 = (10 + 512 + 512^2 +1) data blocks +
1 (single) + 1 + 512 (double) + 1 + 1 + 1 (triple) index blocks
= 16 + 1024 + 512^2 = 1040 + 512^2

N1 = floor(200M / 12) = 17476266 » 16,66 M
N3 = floor(200M / (1040 + 512^2)) = 796

B) Given a binary file of size 20490.5KB, compute exactly how many index blocks and data blocks the file is using.

Data blocks: ceil(20490.5KB/2KB) = 10246
Int. frag = 1-0.25 blocks = 0.75 blocks = (1024+512)B = 1536B
Index blocks
Single index: 1
Inner index blocks (double): ceil((10246-10-512)/512) = 19
Outer index block (double): 1 (double indirect is enough)
Total index blocks: 1+19+1 = 21

C) Consider the same file of Q B, where the lseek(fd,offset,SEEK_SET) operation is called to position the file offset for the subsequent read/write operation. Suppose fd is the file descriptor associated to the file (already open), that offset = 0x008000010. Compute the logical block number (relative to the file blocks, numbered starting at 0) at which the position is moved. If the file uses a single (or double/triple) indirect indexing, compute which row of the outer index block contains the data block index (or the inner index block index).

Data block index: 0x00800010/2K = (8M+2)/2K = 4K = 0x1000
10 data blocks are direct
512 data blocks are at single indirect
The block is at the double indirect level
Outer index = (4K-512-10)/512 = 6

Domanda 3

Risposta non data

Punteggio max.: 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

Answer the following questions on memory management:

A) Consider disk scheduling algorithms. What do the scan and C-scan algorithm optimize and what disk technology do they address? (motivate)

B) What is ***noop*** scheduling, and why is it adopted with NVM (Non-Volatile Memory) disks?

C) Consider a HDD disk with maximum rotational latency (time for one entire disk rotation) 6ms, average seek time 4ms, a 0.2ms controller overhead, and 2Gbit/s transfer rate. Suppose disk blocks have size 8KB, compute the rotational speed (measured in rpm, i.e. revolutions per minute), the average rotational latency (consider half a rotation), the transfer time for one block, and the average IO time to transfer 2 adjacent blocks (in one single IO operation)

A) Consider disk scheduling algorithms. What do the scan and C-scan algorithm optimize and what disk technology do they address? (motivate)

Also known as the "elevator" algorithms, they minimise seek time, and the related seek distance, which means the overall distance traveled by the disk head when locating the requested cylinder. The distance is proportional to the sum of the differences between couples on subsequent disk block indexes (or cylinder indexes) serviced.
In scan, disk requests are serviced in both directions (up and down), whereas c-scan services requests in just one direction.
The algorithms just works with HDDs (magnetic disks), where the seek overhead is related to the seek distance. SDDs do not have seek time.

B) What is ***noop*** scheduling, and why is it adopted with NVM (Non-Volatile Memory) disks?

NOOP scheduling means no operation or no scheduling, so requests can be serviced on a pure FIFO strategy. It is adopted with NVMs, as they do not have seek time (nor rotational latency).

C) Consider a HDD disk with rotational latency (time for one entire disk rotation) 3ms, average seek time 4ms, a 0.2ms controller overhead, and 2Gbit/s transfer rate.

Suppose disk blocks have size 8KB, compute the rotational speed (measured in rpm, i.e. revolutions per minute), the average rotational latency (consider half a rotation), the transfer time for one block, and the average IO time to transfer 2 adjacent blocks (in one single IO operation)

*Rot. Sp.: 60s * 1/6ms = 10000 rpm*
Avg. Rot. Lat = 6ms / 2 = 3ms ***(There was a mismatch between the texts of the question: in considered correct both 6ms and 3ms)***
*Transfer Time (1 block) = 8KB / (2Gbit/s) = (8KB * 8 / 2GB) s = 32 + 10^{-6} s = 0.032 ms*

*Avg. IO time (2 blocks) = Avg. rot. lat. + avg seek + transfer time + controller ovhd = 3ms + 4ms + 2*0.032ms + 0.2ms = 7.264ms*

Domanda 4

Risposta non data

Punteggio max.: 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 console management in OS161, and the functions written below.

```
void putch_intr(struct con_softc *cs, int ch) {
  P(cs->cs_wsem);
  cs->cs_send(cs->cs_devdata,ch);
}

void con_start(void *vcs) {
  struct con_softc *cs = vcs;
  V(cs->cs_wsem);
}
```

Answer the following questions

A) Which one of the two functions is (directly) called by the **putch**(int ch) console function, and which one is called upon receiving an interrupt by the serial console (motivate)?

B) Provide the definition of the cs_wsem field in struct con_softc

C) Provide an implementation of the two functions, by replacing the cs_wsem semaphore with a condition variable (also define the required field(s) in struct con_softc)

A) Which one of the two functions is (directly) called by the **putch**(int ch) console function, and which one is called upon receiving an interrupt by the serial console (motivate)?

putch_intr is called by the putch function, as it receives the char to be printed, waits for the serial console to be ready (`P(cs->cs_wsem);`), then calls the send (cs_send) function of the console device driver.
Whereas con_start is activated by the hardware interrupt related to the serial console that means “console ready to receive a new output”, so the call to `V(cs->cs_wsem);` signals the waiting putch_intr that a new output can be done.

B) Provide the definition of the cs_wsem field in struct con_softc

```
struct con_softc {
  ...
  struct semaphore *cs_wsem;
  ...
}
```

C) Provide an implementation of the two functions, by replacing the cs_wsem semaphore with a condition variable (also define the required field(s) in struct con_softc)

We need a condition variable and a lock.
Let's consider that multiple concurrent calls to putch could be done. Just one at a time will be released by the semaphore. As an alternative solution, we use a "ready" flag, a lock for mutual exclusion and the condition variable for wait/signal.

```
struct con_softc {
  ...
  struct lock *cs_wlk;
  int wready;
  struct lock *cs_wcv;
  ...
}

void putch_intr(struct con_softc *cs, int ch) {
  /* wait for the console to be ready */
  lock_acquire(cs->cs_wlk;
  while (!cs->wready) {
    cs_wait(cs->cs_wcv, cs->cs_wlk);
  }
  cs->wready = 0;
  lock_release(cs->cs_wlk;
  /* release lock, wready will be set by con_start after the
output is done */
  cs->cs_send(cs->cs_devdata,ch);
}

void con_start(void *vcs) {
  struct con_softc *cs = vcs;
  /* set the ready flag and wake up a thread possibly waiting
*/
  lock_acquire(cs->cs_wlk;
  cs->wready = 1;
  cs_signal(cs->cs_wcv,cs->wlk);
  lock_release(cs->cs_wlk;
}
```

Domanda 5

Risposta non data

Punteggio max.: 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_switch?

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?

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_switch?

Both structures are used to save a process context (registers + other info): the switchframe is used for a context switch (a change of running process 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 in order to start a user process with mips_usermode.
A system call exploits the trapframe, as it is activated as a trap. The thread_switch function uses a switch frame for the new thread to be queued in the ready queue.

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

In the kernel-level stack, as 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 need to be mapper in the 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 need to 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?

NO, for lack of privilege. The buf pointer is the source of the write operation, whereas the destination is the file, which could be either e regular 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.