System and device programming

Iniziato	giovedì, 10 giugno 2021, 15:26
Stato	Completato
Terminato	giovedì, 10 giugno 2021, 15:26
Tempo impiegato	6 secondi
Valutazione	**0,00** su un massimo di 15,00 (**0**%)

Domanda 1

Risposta non data

Punteggio max.: 3,00

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

Consider a virtual memory system based on paging, with a Byte addressable RAM. The system has a TLB (Translation Look-aside Buffer). A two level page-table is used, based on splitting a 64-bit logical address (from MSB to LSB) into 3 parts: *p1*, *p2* and *d:* p2 uses14 bits, and d uses 13 bits. No other data structures (such as hash tables or inverted page tables) are used. Virtual memory is managed with demand paging.

Answer the following questions:

A) Assuming the RAM memory has an access time T_{RAM} = 160 ns, calculate the TLB miss ratio needed in order to guarantee an effective access time EAT_{PT} <= 200 ns, assuming that the TLB lookup time is negligible.
Is the result a lower bound or an upper bound for the TLB miss ratio?

B) Now consider the page fault frequency p_{PF}. An experimental evaluation shows that a page fault is served in 4 ms on average, and that the performance decrease (increase in running time) due to page faults, is between 10% and 20%.
Compute the range of values for p_{PF} that are compatible with the experimental evaluation.

C) Another experimental evaluation/simulation has been done using two reference strings w_1, w_2, having length, respectively, $len(w_1)=10^6$, $len(w_2)=2*10^6$, and probability p_1 and p_2. The simulation generated 500 page faults overall (100 with w_1 and 400 with w_2) and estimated an empirical probability f = 0.00013 (expected frequency) for a page fault occurring in the real system. Compute the values of p_1 and p_2

A) TLB miss ratio = ...
upper or lower bound?

T_{RAM} = 160 ns (RAM access time)
EAT_{PT} = 200 ns
$x = miss_{TLB} = 1 - h_{TLB}$
2 level (hierarchical) PT => 2 reads for PT lookup

$EAT_{PT} = h_{TLB}*T_{RAM} + (1-h_{TLB})*3T_{RAM} = (1 + 2*(1-h_{TLB}))T_{RAM} = (1 + 2x)\ T_{RAM}$
$2x = EAT_{PT}/T_{RAM} - 1 = 1/4$
$x = 1/8 = 0.125$
Upper bound as a higher miss ratio increases EAT_{PT}

B) Range of values for p_{PF}

T_{PF} = 4 ms (PF service time)

$p_{PF}*T_{PF}$ = increase in time due to page faults

$0.1 < p_{PF}*T_P/ EAT_{PT} < 0.2$

$0.1 < p_{PF}*(4/ 200)*10^6 < 0.2$

$0.1 < 2*10^4 p_{PF} < 0.2$

$5*10^{-6} < p_{PF} < 10^{-5}$

$p_{min} = 5*10^{-6}$

$p_{Max} = 10^{-5}$

C) Calculation of string probabilities $p_1$and p_2

F_1=100, F2=400 (F_1+F_2 = 500)

p1 + p2 = 1 (we just have two strings)

$f = p_1*F_1/len(w_1) + p_2*F_2/len(w_2)$

$1.3*10^{-4} = p_1*100/10^6 + p_2*400/(2*10^6)$

$p_1 = 1 - p_2$

$(1-p_2)*10^{-4} + 2p_2*10^{-4} = 1.3*10^{-4}$

$1+p_2 = 1.3$

$p_2 = 0.3$

$p_1 = 0.7$

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

A given disk is organised with physical and logical blocks of given (same) size BS. The disk contains multiple partitions: partition A is formatted for a file system that statically allocates NM=12.5K blocks for metadata (directories, file control blocks and a bitmap, for free space management), and ND=100M blocks for file data.

Answer the following questions:

A) Compute the block size BS, considering that NM/4 blocks are reserved to the bitmap, that has one bit for each of the ND data blocks.

B) We know that a file control block (FCB) has size 512B and NM/2 metadata blocks are reserved to FCBs, compute the maximum number of files that can be stored in the file system.

C) Suppose the bitmap indicates a 50% ratio free/used blocks (so 1 free block for 2 used, used means "allocated"), and that exactly 5M (M = 2^{20}) free blocks are "isolated" (preceded and followed by an allocated block), compute the maximum size for a contiguous interval of free blocks, assuming the most favorable bitmap configuration. Give the same answer when assuming the less favorable bitmap configuration.

A) Compute the block size BS, considering that NM/4 blocks are reserved to the bitmap, that has one bit for each of the ND data blocks.

The bitmap size can be expressed both given the total mount of bits (ND) and the number of blocks reserved to the bitmap (NM/4)
|bitmap| = ND/8 Bytes = NM/4 * BS
BS = ND/(2*NM) Bytes = 100M/25K Bytes = 4KBytes

B) We know that a file control block (FCB) has size 512B and NM/2 metadata blocks are reserved to FCBs. Compute the maximum number of files that can be stored in the file system.

The maximum number of files corresponds to a full occupancy of the FCB reserved space
NF_{Max} * |FCB| = NM/2
Let's express |FCB| in terms of blocks (we could work on Bytes as well): |FCB| = 1/8 Block
NF_{Max} = NM/(2*|FCB|) = 12.5K/(2/8) = 4*12.5K = 50K

C) Compute the size of the largest free block interval |LargestFree| in the most favourable and in the least favourable scenarios (sizes have to be expressed as numbers of blocks):

the bitmap has ND bits, corresponding to ND blocks
ND/3 blocks are free (1 free, 2 used means 1 free every 3 blocks), 5M free blocks are "isolated"

Most favourable:
All non "isolated free blocks are contiguous, so the largest interval on contiguous free blocks is given by
|LargestFree| = ND/3 - 1M blocks.
ND = 100M
|LargestFree| = 33.3M - 5M blocks = 28.3M blocks

Least favourable
The least favourable situation is when all non "isolated" free blocks are grouped by couples of 2 contiguous free blocks interleaved with allocated blocks.
|LargestFree| = 2 blocks

Domanda 3

Risposta non data

Punteggio max.: 3,00

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

Let us consider optimizations implemented in order to improve the overall performance of virtual memory management.

A) Briefly explain the COW (Copy on Write) technique: what is it and how does it improve memory management performance? Is the expected gain/improvement on time? On memory size? On both time and memory? (Motivate)

B) Explain why the IO on a swap partition (backing store) is faster than file system IO (even if on the same device). Can the virtual address space of a given process be larger than the size of the swap area (size of the swap partition or file)? (Motivate)

C) What is the role of the modify (dirty) bit associated to a page? Is the modify (dirty) bit just stored in the page table or is it also stored in the TLB (when available in a given processor)? (Motivate)

A) Briefly explain the COW (Copy on Write) technique: what is it and how does it improve memory management performance? Is the expected gain/improvement on time? On memory size? On both time and memory? (Motivate)

- COW is an optimisation allowing two processes (parent and child) to share their pages upon child creation. Pages are duplicated (so they become two independent pages) when either process (parent or child) modifies them by writing.
- Performance is improved both on time and memory: time, because the process creation is faster as pages are initially shared (part of the time is simply deferred to a later writing, but just for written pages), memory, because new pages are generated just if/when written

B) Explain why the IO on a swap partition (backing store) is faster than file system IO (even if on the same device). Can the virtual address space of a given process be larger than the size of the swap area (size of the swap partition or file)? (Motivate)

It is faster because the swap space is allocated in larger chunks and less management is needed than file system.
Yes it can be larger, as the swap space just need to provide storage for the used pages, not foe the unused ones (think for instance of the possibly unused pages in the "middle" of the address space.

C) What is the role of the modify (dirty) bit associated to a page? Is the modify (dirty) bit just stored in the page table or is it also stored in the TLB (when available in a given processor)? (Motivate)

The modify bit keeps track of the state of a page with respect of its copy on disk, or with respect to a given time interval of observation.
Though the bit is stored in each page table entry, storing it also within each TLB entry allows best performance (no PT access is due with TLB hit), but requires hardware support, in order to set the bit when writing the page.

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 three OS161 kernel threads implementing a data transfer task based on a producer/consumer model (1 producer, 2 consumers). The threads share a data buffer, implemented as a C structure of type `struct prodConsBuf` defined as follows:

```
struct prodConsBuf {
  data_t data;
  int dataReady;
  struct lock *pc_lk;
  struct cv *pc_cv;
};
```

Whenever no data is present in the buffer, the dataReady flag is 0.
When the producer writes new data (pointed by dp) to the buffer (data field), he sets dataReady at 1 and signals both consumers (the work is done by calling function producerWrite).

```
void producerWrite(struct prodConsBuf *pc, data_t *dp);
```

Consumers iteratively execute the consumerRead function.

```
void consumerRead(struct prodConsBuf *pc, data_t *dp);
```

When signalled, just one consumer is able to find the dataReady flag set (value 1). He then actually reads the data (from the buffer to the memory pointed by dp), clears the flag, and returns from the consumerRead function. The consumer finding the dataReady flag false (value 0) simply keeps on waiting for another signal.
The lock is used for mutual exclusion. The condition variable is used for producer-to-consumer signals. Just consider producer-to-consumer synchronization: THE PRODUCER IS NOT REQUIRED TO CHECK IF PREVIOUS DATA WERE READ. We can safely assume that a new call to `producerWrite` is done after all previous data were read by consumers.

Answer the following questions:

A) Can the shared structure be located into the thread stack, or should it be a global variable or other?

B) As the `pc_lk` and `pc_cv` fields are pointers, where should the `lock_create()` and `cv_create()` be called? In the producer thread, in the consumer threads? Elsewhere?

C) Provide an implementation of functions `producerWrite` and `consumerRead`. No other producer and/or consumer code is needed, just the functions.

A) Can the shared structure be located into the thread stack, or should it be a global variable or other?

Each thread has its own thread stack, so a shared data cannot reside there. A global variable is the usual option. Other options include dynamic allocation (by kmalloc), if properly handled.

B) As the `pc_lk` and `pc_cv` fields are pointers, where should the `lock_create()` and `cv_create()` be called? In the producer thread, in the consumer threads? Elsewhere?

It could be done in each of them, provided that they properly synchronize: e.g. one thread does initializations, the other threads wait.
But a more common solution could be that initializations are done by another thread, the parent/master thread, who is creating the producer and the consumers.

C) Provide an implementation of functions `producerWrite` and `consumerRead`. No other producer and/or consumer code is needed, just the functions.

```
/* error checking/handling instructions are omitted for
simplicity */

void producerWrite(struct prodConsBuf *pc, data_t *dp) {
  /* lock for mutual exclusion */
  lock_acquire(pc->pc_lk);
  /* copy data: done here by simple assignment -
     a proper function could be used instead */
  pc->data = *dp;
  pc->dataReady = 1; /* set flag */
  /* signal all consumers */
  cv_broadcast(pc->pc_cv,pc->pc_lk);
  /* just release the lock and return */
  lock_release(pc->pc_lk);
}

void consumerRead(struct prodConsBuf *pc, data_t *dp) {
  /* lock for mutual exclusion */
  lock_acquire(pc->pc_lk);
  while (!pc->dataReady) {
    cv_wait(pc->pc_cv, pc->pc_lk);
  }
  /* copy data: done here by simple assignment -
     a proper function could be used instead */
  *dp = pc->data;
  pc->dataReady = 0; /* reset set flag */
  /* just release the lock and return */
  lock_release(pc->pc_lk);
}
```

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 possible implementation of the file system calls in OS161, based on per-process and system file tables defined as follows

```
/* system open file table */

struct openfile {

  struct vnode *vn;

  off_t offset;

  unsigned int countRef;

};

/* this is a global variable */

struct openfile systemFileTable[SYSTEM_OPEN_MAX];

...

/* user open file table: this a field of struct proc */

struct openfile *fileTable[OPEN_MAX];
```

Answer the following questions (all answers have to be motivated):

A) Why is `systemFileTable` a global variable, whereas `fileTable` is a field of `struct proc` ?

B) Suppose two user processes call open() for the same file, will there be two entries in the `systemFileTable` for the file, or just one entry (pointed to, thus shared, by the two processes) ?

C) We need to implement the support for the SYS_lseek system call. The lseek function is described as follows:

`off_t lseek(int fd, off_t offset, int whence);`

`lseek()` repositions the file offset of the open file description associated with the file descriptor `fd` to the argument offset according to the directive `whence` as follows:

SEEK_SET

The file offset is set to offset bytes.

SEEK_CUR

The file offset is set to its current location plus offset bytes.

SEEK_END

The file offset is set to the size of the file plus offset bytes.

Which data structures will be affected by the implementation of the SYS_lseek system call?

1. the process file table?
2. one entry of the systemFileTable (for a given call to lseek)?
3. multiple entries of the systemFileTable (for a given call to lseek)?

4. the vnode of the file?

(multiple selections are possible, for each selected option explain what is done on the data structure)

A) Why is `systemFileTable` a global variable, whereas `fileTable` is a field of `struct proc` ?

`systemFileTable` a global variable because it is unique and shared by all processes, whereas we have one `fileTable` for each process, so we can directly allocate it within the `struct proc`.

B) Suppose two user processes call open() for the same file, will there be two entries in the `systemFileTable` for the file, or just one entry (pointed to, thus shared, by the two processes) ?

There will be two entries, because each of them needs a different offset within the file (two processes read/write at different locations within the file), so a single entry cannot be shared. A single `struct openfile` (an entry in the `systemFileTable`) can be shared only as a result of operations such as the dup and dup2 system calls..

C) We need to implement the support for the SYS_lseek system call. Which data structures will be affected by the implementation of the SYS_lseek system call?

1. the process file table? NO
2. one entry of the systemFileTable (for a given call to lseek)? YES
3. multiple entries of the systemFileTable (for a given call to lseek)? NO
4. the vnode of the file? NO

Just one openfile entry within the systemFileTable will be modified in order to update its offset field, as the lseek operation just modifies the offset for one process (same as with read/write operations.