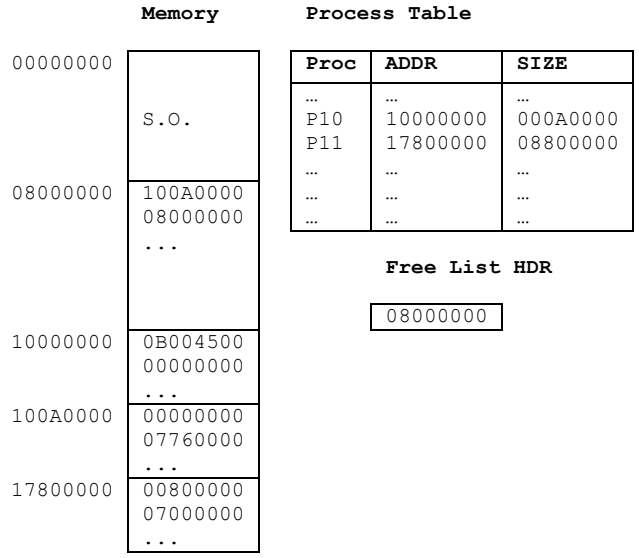
Exercises from old exams

Memory Management 1 (exercises on ch 9)

1. *(5/7/2011)* Let's have a system equipped with a 512MByte RAM, where proces allocation is handled following a variable (contiguous) partition scheme, with allocation done on multiples of 64 bytes.
   The Operating System is resident (permanently) in the first RAM section of 128MBytes.
   The process table contains, for each active process, the start address (ADDR) and the size (SIZE) of the associated memory partition. Memory partitions are allocated with a Worst-Fit strategy. Free partitions are handled using a free list, a linked list of free partitions, sorted by decreasing size (so that the first partition is the largest one); a list node had two fields: **pointer to the next partition**, **partition size.** Both are represented on 4 bytes (with value 0 used as NULL pointer**)** and stored in the first 8 bytes of the partition they represent.

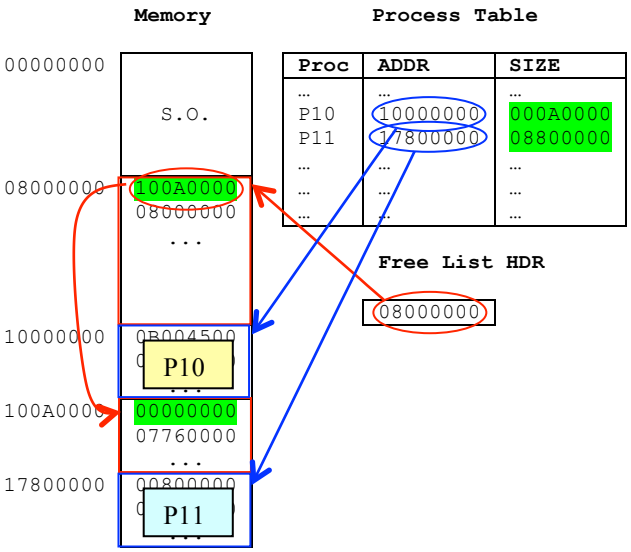


The figure shows the (partial) contents of the process table and of the free list header at a given time. A partial content of memory partitions is also shown.

Q: show the changes to be made on partitions, partition table and free list header after 2 new processes are started: P12 and P13, requiring 25MB and 150MB, respectivel. Then P11 terminates.

PROPOSED SOLUTION

Initial configuration



Initial configuration, explicty showing pointers and partition sizes.

Notice that partitions in the free list start with size and pointer to the next element in the list. The content of allocated partitions is not meaningful to this problem, as it is process-dependent (the content depends on the process itself, this is why partitions are "covered" by a rectangle).

In a more compact form, the initial configuration can be represented as follows

Memory is split in in 5 partitions. The first one is allocated to the Operating System, the other 4 partitions are interleaved as follows: P10, free, P11, free. P10 and P11 are referred by the Partition Table, the other partitions are linked in the free list. Partitions in the free list have, in their first 8 Bytes, a couple (pointer, size). In allocated partition, the initial 8 bytes are not meaningful to the problem.

We suggest to keep addresses and sizes in hexadecimal, as easier to manipulate. The initial partitions, listed by couples (start address, size) are:

Hex
(00000000,08000000), (08000000,08000000), (10000000,000A0000), (100A00000,07760000), (17800000,08800000)

Grouped by type

OS: (00000000,08000000)
Free list: (08000000,08000000), (100A0000,07760000)
P10: (10000000,000A0000)
P11: (17800000,08800000)

Allocating P12.

We need 25MB (in hex code: 19MB = 01900000 B). Both free partitions have greater size, the larger one (the first) is (partially) allocated: let's suppose we allocate the needed space on lower addresses, then the partition leaves a free leftover (of size 08000000 B – 01900000 B = 06700000 B). La newly (reduced) free partition goes to the end of the free list (as the list is sorted by decreasing size). Changes w.r.t. the inititial configuration are represented, only.

Free list: (100A0000,07760000), (09900000,06700000)
P12: (08000000,01900000)

Allocating P13.

P13 needs 150MB (in hex code: 96MB = 09600000 B). None of the two free partitions is large enough. Multiple solutions are possible, such as de-fragmentation (with process relocation and memory compaction), swap-out of P10 or P11, in order to free enough space for P13, or waiting, i.e. deferring the start of P13. Let us represent a swap-out solution: swapping-out P10 allows to create a free partition of size 0DF00000 B (dagiven by 06700000+000A0000+07760000), where P13 can be allocated in the initial part (with the leftover generating a free partition).

Free list: (12F000000,04900000)
P13: (09900000,09600000)

Termination of P11.

The P11 partition is freed, thus increasing the size of the single free partition in the free list.

Free list: (12F000000,0D100000)

In case of P10 sapped-out, it could be resumed …

2. *(1/9/2008)* Let's have a system with virtual (Byte addressable) memory management based on paging. The MMU has a TLB (Translation Look-aside Buffer), on which an expemental evaluation have measured a 99 % "hit ratio". The Page Table has adopts a two-level scheme (hyerarchical PT), where a 32 bit logic address is split (from MSB to LSB) in 3 parts: p1, p2, d, of 10 bits, 11 bits, 11 bits, respectively. No hashing or inverted page table are used to speed up memory accesses.

   Q.
   - What do we mean by "hit ratio"?
   - Show the PT scheme and evaluate it's overall size, for a P1 process characterized by a logic address space of 100 MBytes.

- Compute the external and internal of process P1 (the same proces as in the previous question).
- Suppose the RAM memory has access time 300 ns, compute the effective access time (EAT) for the proposed case study (hit ratio = 99 %)

A	• (see slides/book) the hit ratio is the ratio of TLB successfull translations divided by the total number of memory accesses. • Consider that 100 MBytes < 128 MBytes = 2^{27} Bytes and that pages/frames have size 2Kbytes = 2^{11} Bytes (given by d: 11 bit). Let's uppose every PT entry uses 32 bits (4 Bytes), one of the second level (inner) PTs has 2^{11} entries (11 bit p2) and size 2^{11}*4 Bytes, whereas the first level (outer) PT has 2^5 entries (just 5 bits of p1, out of 10 are needed) and size di 2^5*4 Bytes. The number of inner PTs is 25, so the outer PT has some internal fragmentation (a solution with an outer PT of just 25 entries could be acceptable as well). The figure/representation is left to the student. • External fragmentation is 0 by definition. Internal fragmentation, for an arbitrary process, would be: o worst case: 2 Kbyte (over-approximation of 2K-1) o average case: 1 Kbyte we exactly know the size of P1, so its internal fragmentation can be computed in an exact way: 0, as the logical address space is a multiple of 2Kbytes. • EAT = (0.99 * 300 + 0.01 * 3 * 300) ns = 1.02 * 300ns = 306 ns WARNIG: in case of TLB MISS, we need 3 memoty accesses (not as in the slides), because of the two-level PT, that needs 2 reads for page-to-frame translation, plus the final phisical RAM access

2. *(25/6/2018)* Briefly describe advantages and disadvantages of an inverted page table (IPT), with respect to a standard PT (possibly hyerarchical). Let's consider a process with a 32GB logic address space, on a 64 bits (Byte addressable) system with a 8GB RAM, paging enabled, with 1KB page/frame size. Compare a solution based on a standard PT (one PT for each process) and a solution based on IPT. Compute the sizes of the (single level) PT and of the IPT for the given process. If needed, represent thr process ID (`pid`) on 16 bits. Use 32 bits for page and/or frame numbers (indexes). Also compute the maximum possible size for the virtual/logic address space of a process, using the given IPT.

A	***Advantages*** • Saving memory: the IPT size is based on the phisical RAM, rather than on the logic/virtual address space • Unique table for all processes (a frame is owned by a single processa t any given time) **Disadvantages** • Time: the IPT can be slow, as a page index has to be searched, rather than used for direct/random O(1) reference. For the above reason IPTs are typically associated with hashing. **When computing sizes, we omit validity/modify bits.** **Standard Page Table:** N pages = 32GB/1KB = 32M (number of frame indexes/numbers in the PT) |Page Table| = 32M*4B = 128MB **IPT** The IPT, unique and shared by all processes, has fixed size, as every entry is associated to a RAM frame (let's consider, for simplicity, that the whole RAM is addressed (which is not true in general, and the part reserved to the OS is not available to processes). Each IPT entry contains a page number (4B) and a pid (2B) N frame = 8GB/1KB = 8M |IPT| = 8M*(4B+2B) = 48MB **Virtual/logic address space** The maximum number of process pages is limited by the size (32 bits) of page indexes (NOT by the number of IPT entries!): the number is 4G. A the page size is 1KB, the virtual address space has maximum size 4G*1KB = 4TB.