

CEG 433/633 Operating Systems  
 Match

Dynamic Storage Allocation: Buddy System

C. The "buddy system." We will now describe another approach to dynamic storage allocation, suitable for use with binary computers. This method takes one bit of "overhead" in each block, and it requires all blocks to be of length 1, 2, 4, 8, or 16, etc. If a block is not  $2^k$  words long for some integer  $k$ , the next higher power of 2 is chosen and extra unused space is allocated accordingly. When this method is applicable it has an advantage of speed, especially in "real-time" situations, since there is a positive guarantee against searches through a long AVAIL list.

The idea of this method is to keep separate lists of available blocks of each size  $2^k$ ,  $0 \leq k \leq m$ . The entire pool of memory space under allocation consists of  $2^m$  words, which we will assume for convenience have the addresses 0 through  $2^m - 1$ . Originally, the entire block of  $2^m$  words is available. Later, when a block of  $2^k$  words is desired, and if none of this size are available, a larger available block is split into two equal parts; ultimately, a block of the right size  $2^k$  will appear. When one block splits into two (each of which is half as large as the original), these two blocks are called *buddies*. Later when both buddies are available again, they coalesce back into a single block; thus the process can be maintained indefinitely (unless we run out of space at some point).

The key fact underlying the practical usefulness of this method is that if we know the address of a block (i.e., the memory location of its first word), and if we also know the size of that block, we know the address of its buddy. For example, the buddy of the block of size 16 beginning in binary location 10111001010000 is a block starting in binary location 101110010100000. To see why this must be true, we first observe that as the algorithm proceeds, the address of a block of size  $2^k$  is a multiple of  $2^k$ . In other words, the address in binary notation has at least  $k$  zeros at the right. This observation is easily justified by induction: if it is true for all blocks of size  $2^{k+1}$ , it is certainly true when such a block is halved.

Therefore a block of size, say, 32 has an address of the form  $xx \dots x00000$  (where the  $x$ 's represent either 0 or 1); if it is split, the newly formed buddy blocks have the addresses  $xx \dots x00000$  and  $xx \dots x10000$ . In general, let  $\text{buddy}_k(x) =$  address of the buddy of the block of size  $2^k$  whose address is  $x$ ; we find that

$$\text{buddy}_k(x) = \begin{cases} x + 2^k, & \text{if } x \bmod 2^{k+1} = 0; \\ x - 2^k, & \text{if } x \bmod 2^{k+1} = 2^k. \end{cases} \quad (10)$$

From: Donald Knuth, "The Art of Computer Programming, Vol 1: Fundamental Algorithms," Addison-Wesley.

This function is readily computed with the "exclusive or" instruction (sometimes called "selective complement" or "add without carry") usually found on binary computers; cf. exercise 28.

The buddy system makes use of a one-bit TAG field in each block:

$$\begin{aligned} \text{TAG}(P) &= 0, & \text{if the block with address } P \text{ is reserved;} \\ \text{TAG}(P) &= 1, & \text{if the block with address } P \text{ is available.} \end{aligned} \quad (11)$$

Besides this TAG field, which is present in all blocks, available blocks also have two link fields, LINKF and LINKB, which are the usual forward and backward links of a doubly linked list; and they also have a KVAL field to specify  $k$  when their size is  $2^k$ . The algorithms below make use of the table locations AVAIL[0], AVAIL[1], ..., AVAIL[ $m$ ], which serve respectively as the heads of the lists of available storage of sizes 1, 2, 4, ...,  $2^m$ . These lists are doubly linked, so as usual (see Section 2.2.5) the list heads contain two pointers:

$$\begin{aligned} \text{AVAILF}[k] &= \text{LINKF}(\text{LOC}(\text{AVAIL}[k])) = \text{link to rear of AVAIL}[k] \text{ list;} \\ \text{AVAILB}[k] &= \text{LINKB}(\text{LOC}(\text{AVAIL}[k])) = \text{link to front of AVAIL}[k] \text{ list.} \end{aligned} \quad (12)$$

Initially, before any storage has been allocated, we have

$$\begin{aligned} \text{AVAILF}[m] &= \text{AVAILB}[m] = 0, \\ \text{LINKF}(0) &= \text{LINKB}(0) = \text{LOC}(\text{AVAIL}[m]), \\ \text{TAG}(0) &= 1, \text{KVAL}(0) = m \end{aligned} \quad (13)$$

(indicating a single available block of length  $2^m$ , beginning in location 0), and also

$$\text{AVAILF}[k] = \text{AVAILB}[k] = \text{LOC}(\text{AVAIL}[k]), \quad \text{for } 0 \leq k < m \quad (14)$$

(indicating empty lists for available blocks of lengths  $2^k$  for all  $k < m$ ).

From this description of the buddy system, the reader may find it enjoyable to design the necessary algorithms for reserving and freeing storage areas by himself, and to compare his solutions with the algorithms given below. Note the comparative ease with which blocks can be halved in the reservation algorithm.

CEG 433/633

Matsuki

**Algorithm R** (*Buddy system reservation*). This algorithm finds and reserves a block of  $2^k$  locations, or reports failure, using the organization of the buddy system as explained above.

**R1.** [Find block.] Let  $j$  be the smallest integer in the range  $k \leq j \leq m$  for which  $AVAILF[j] \neq LOC(AVAIL[j])$ , that is, for which the list of available blocks of size  $2^j$  is not empty. If no such  $j$  exists, the algorithm terminates unsuccessfully, since there are no known available blocks of sufficient size to meet the request.

**R2.** [Remove from list.] Set

$$L \leftarrow AVAILF[j], \quad AVAILF[j] \leftarrow LINKF(L), \\ LINKB(LINKF(L)) \leftarrow LOC(AVAIL[j]), \quad \text{and} \quad TAG(L) \leftarrow 0.$$

**R3.** [Split required?] If  $j = k$ , the algorithm terminates (we have found and reserved an available block starting at address  $L$ ).

**R4.** [Split.] Decrease  $j$  by 1. Then set

$$P \leftarrow L + 2^j, \quad TAG(P) \leftarrow 1, \quad KVAL(P) \leftarrow j, \quad LINKF(P) \leftarrow LOC(AVAIL[j]), \\ LINKB(P) \leftarrow LOC(AVAIL[j]), \quad AVAILF[j] \leftarrow AVAILB[j] \leftarrow P.$$

(This splits a large block and enters the unused half in the  $AVAIL[j]$  list which was empty.) Go back to step R3. ■

( $malloc(x), x == 2^k$ )

**Algorithm S** (*Buddy system liberation*). This algorithm returns a block of  $2^k$  locations starting in address  $L$  to free storage, using the organization of the buddy system as explained above.

**S1.** [Is buddy available?] Set  $P \leftarrow buddy_k(L)$ . (See Eq. 10.) If  $k = m$  or if  $TAG(P) = 0$ , or if  $TAG(P) = 1$  and  $KVAL(P) \neq k$ , go to S3.

**S2.** [Combine with buddy.] Set

$$LINKF(LINKB(P)) \leftarrow LINKF(P), \quad LINKB(LINKF(P)) \leftarrow LINKB(P).$$

(This removes block  $P$  from the  $AVAIL[k]$  list.) Then set  $k \leftarrow k + 1$ , and if  $P < L$  set  $L \leftarrow P$ . Return to S1.

**S3.** [Put on list.] Set

$$TAG(L) \leftarrow 1, \quad LINKF(L) \leftarrow AVAILF[k], \quad LINKB(AVAILF[k]) \leftarrow L, \\ KVAL(L) \leftarrow k, \quad LINKB(L) \leftarrow LOC(AVAIL[k]), \quad AVAILF[k] \leftarrow L.$$

(This puts block  $L$  on the  $AVAIL[k]$  list.) ■

(free(L))