A Simple Hardware Buddy System Memory Allocator

EWALD VON PUTTKAMER

***Abstract*—The design of a simple hardware memory allocator is described, which allocates blocks of different lengths $L = 2^h$, $h = K$, K-1,$\cdots$,K-n in a memory according to the buddy system algorithm. The binary tree, representing the distribution of free and used blocks in memory is mapped into a set of shift registers. They are connected for end-around shifting and clocked with frequencies different for each register, preserving thereby the internal structure of the binary tree. A small counter, attached to each shift register in the set holds the address of the first free block and can be read out on request within 0.5 μs. A simple control unit realizes the algorithm. Having answered a request the system needs about 130 μs to compute the addresses of free blocks in a total of 511 blocks managed by the device.**

***Index Terms*—Binary tree representations, buddy system representations, hardware: substitution of software, operating systems: memory allocation.**

INTRODUCTION

MEMORY allocation is one of the major tasks of the memory management code in any operating system. Blocks of data and programs of various sizes may have to be swapped between core memory and mass storage. The memory manager has to find free space in storage in blocks of prescribed length and has to keep track of blocks of store no longer in use.

There are various methods of finding free blocks of prescribed length in a store [1]–[3], [7]. They are usually provided as software realizations and use up some time of the central processing unit (CPU). Multiprogramming and timesharing may need many swapping operations as well as dynamic expansion or contraction of programs, and consequently the time spent in memory housekeeping can be an appreciable fraction of the total time used to run a program.

A hardware unit which, when requested for a memory block of prescribed length, will quickly provide a pointer to the next free block and will do its housekeeping "off line" afterwards, will reduce the burden upon the memory manager software and save time. Such a hardware memory allocator has a black box description as shown in Fig. 1. The device must respond to two types of commands from the memory manager.

The first is REQUEST and the second is FREE.

Given a REQUEST with the length of the desired block of storage, the hardware system must answer with the starting address of a free block of at least that size in memory or signal an error if there is no space of at least that size left. When there is an error signal, the memory manager has to take emergency measures like compacting memory or garbage collection. A BUSY signal will show that the system is still housekeeping, i.e., rearranging its internal lists to find the next free block.

The other command is the FREE command. Given the address and the length of a block no longer used, the hardware allocator must register this block as being free and the allocator will be BUSY until internal housekeeping has ended.

The allocator will be organized internally in three main parts:

1) a store for block addresses with flags, indicating which block is free and which one used;

2) a set of pointers to the first free block of any given length; and

3) a control unit governing internal housekeeping.

THE BUDDY SYSTEM

The main determining feature of the hardware unit is the organization of the store for block addresses. To make this organization comparatively simple, we shall use the buddy system algorithm, first described by Knowlton [3]. A serious restriction of this algorithm however is that blocks in the store may be requested and allocated in length of powers of 2 only.

Let there be a store of 2^K words. Block sizes then are $2^K, 2^{K-1}, \cdots, 2^{K-n}$ words. That is, $K = 15$, $n = 8$ describes a 32 768 word memory with a minimum block size of 128 words. There are 256 blocks of 128 words or 128 blocks of 256 words, $\cdots$, or 2 blocks of 16 384 words or 1 block of 32 768 words. In general there are $2^{n+1} - 1$ potential blocks to be handled (one counts all possible memory subdivisions). Fig. 2 shows a store with its various possible blocks for $n = 4$ and a part of store in use. Fig. 3 describes this situation by a binary tree in which the nodes show whether the corresponding part of memory is in use or free. The tree has $n + 1$ levels and in each level h, with $0 \leq h \leq n$, there are 2^h nodes. Within level h each node $T_{h,j}$, with $0 \leq j \leq 2^h - 1$, represents a block of storage of size 2^{K-h} and a starting address $j * 2^{K-h}$ in memory. In Fig. 3 the node $T_{3,3}$ is marked, according to the storage in use in Fig. 2. The other nodes marked show that corresponding blocks are no longer free, being partly occupied by block $T_{3,3}$. Realizing this binary tree as a bit table in memory leads to the allocator described by Isoda *et al.* [4].

Manuscript received February 28, 1974; revised April 11, 1975.
The author is with the Department of Computer Science, University of Kaiserslautern, D675 Kaiserslautern, Germany.

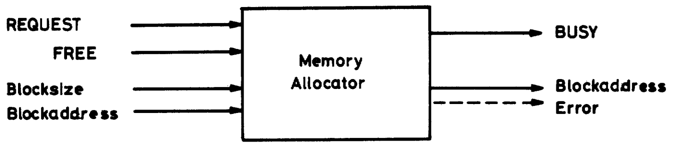

Fig. 1. Black box description of a memory allocator.

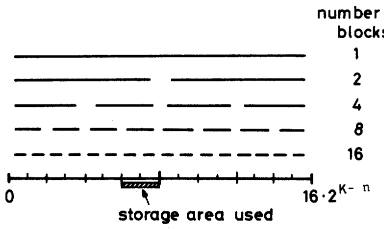

Fig. 2. A store divided into $2^{K-n} = 16$ blocks; note that the total number of blocks in all five levels is $2^{n+1} - 1$.

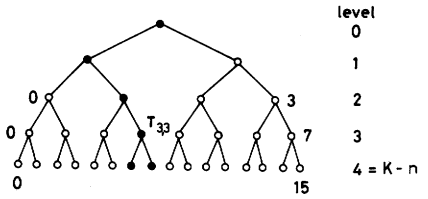

Fig. 3. Binary tree; the darkened nodes show blocks that are in use.

The buddy system algorithm based on a binary tree is as follows: each node $T_{h,j}$ in the tree with $h \neq 0$ has a father node $T_{h-1,\lfloor j/2 \rfloor}$ and for each node $T_{h,j}$ with $h \neq n$ there are two sons $T_{h+1,2j}$ and $T_{h+1,2j+1}$. (The notation $\lfloor a \rfloor$ denotes the greatest integer less than or equal to a [5], [6].) Let node $T_{h,j} = 1$ denote a block in use, while $T_{h,j} = 0$ shows the associated block is free.

Requesting then a free block of storage of length 2^{K-h} means looking into the binary tree at level h for the first node with $T_{h,j} = 0$. The desired starting address then is $j * 2^{K-h}$. If there is no unmarked node on level h then an error signal has to be generated. When a free block has been allocated, the corresponding node $T_{h,j}$ has to be marked and all its predecessors and progeny too. Stated as a program the actions taken are as follows.

```
MARK $(h,j)$
  Step A: $T_{h,j} := 1$.
  Step B: For $i := 0$ step 1 until $h - n$ do (mark progeny)
    for $k = 2^i * j$ step 1 until $2^i * (j + 1) - 1$ do
      $T_{h+i,k} := 1$.
  Step C: For $i = h$ step $-1$ until 1 do (mark predecessors)
    $(T_{i-1,\lfloor j/2 \rfloor} := 1; j := \lfloor j/2 \rfloor)$.
```

To free a storage block of length 2^{K-h} beginning at address $j * 2^{K-h}$ means in the binary tree we must unmark node $T_{h,j}$ and all its progeny and moreover if the *buddy* of $T_{h,j}$ (e.g., the node with the same father $T_{h-1,\lfloor j/2 \rfloor}$) is zero, then unmark the father node $T_{h-1,\lfloor j/2 \rfloor}$ too and check the buddies on level $h - 1$, etc. This may be stated as:

```
UNMARK $(h,j)$
  Step A: $T_{h,j} := 0$.
  Step B: For $i = 0$ step 1 until $n - h$ do (unmark progeny)
    for $k = 2^i * j$ step 1 until $2^i * (j + 1) - 1$ do
      $T_{h+i,k} := 0$.
  Step C: For $i = h$ step $-1$ until 1 do (check the buddies)
    if $j$ mod $2 = 0$ and $T_{i,j+1} = 0$ and $T_{i,j} = 0$, then
      $(T_{i-1,\lfloor j/2 \rfloor} := 0; j := \lfloor j/2 \rfloor)$;
    else if $j$ mod $2 = 1$ and $T_{i,j-1} = 0$ and $T_{i,j} = 0$, then
      $(T_{i-1,\lfloor j/2 \rfloor} := 0; j := \lfloor j/2 \rfloor)$.
```

MAPPING THE BINARY TREE INTO HARDWARE

Realizing the binary tree with flip-flops whose states represent the values of the nodes leads to clumsy and expensive solutions: for the example mentioned one would need 511 flip-flops and a lot of control gates at each flip-flop. A better way is to represent the tree in a set of $n + 1$ shift registers $SR_0,\cdots,SR_n$ of lengths $1,2,\cdots,2^n$ bits, respectively, readily available up to lengths of 2048 bits as integrated MOS–FET circuits with clock frequencies in excess of 4 MHz. When connected for end-around shifting and clocked with frequencies f_n for SR_n, $f_n/2$ for $SR_{n-1},\cdots,f_n/2^n$ for SR_0 this set of shift registers preserves the internal connections between a node $T_{h,j}$ and its predecessors and progeny in a binary tree (Fig. 4).

The clock pulses for SR_n are counted into a "path selector" counter PC of length n. This counter indicates simultaneously all the nodes $T_{h,j}$ that form a path from the root $T_{0,0}$ of the binary tree down to the leaf $T_{n,c}$, c being the content of the counter, as can be seen from Fig. 5: the first h most significant bits of the counter form the ordinal number j of the node $T_{h,j}$, which is one of the predecessors of node $T_{n,c}$. Thus one path at a time is accessible. All its nodes have been shifted to the exit of their respective shift register and their values may be altered simultaneously by the control unit as desired.

POINTERS TO THE NEXT FREE BLOCK IN STORAGE

It is simple to find the next free block in storage. An "index" counter C_h, $h + 1$ bits long, attached to each level h will suffice to find it (Fig. 6).

With the path selector counter at zero, start shifting and counting into each counter C_h with frequency $f_n/2^{n-h}$ until the first zero is shifted to the exit of shift register SR_h, then stop counting. If there is no free block of length 2^{K-h} left in memory, then the index counter on level h will run until its overflow bit is set at which time the path selector counter becomes zero again.

Let the memory manager request a block of length 2^{K-h}. Then the index counter C_h is read into an address register ADR of length $n + 1$ so as to copy the most significant bit of C_h into the most significant bit of the address register, yielding the block address of the first free block on level h, namely (content of ADR) $* 2^{K-n}$. This address is ready now to be fed to the memory manager (Fig. 7). An overflow in the address register signals an error to the control unit.

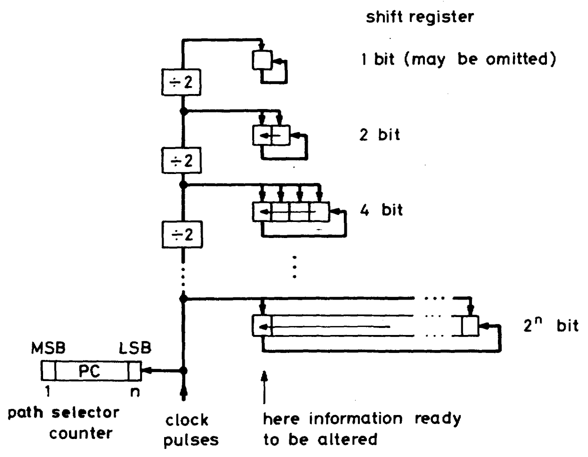

Fig. 4. Tree of shift registers.

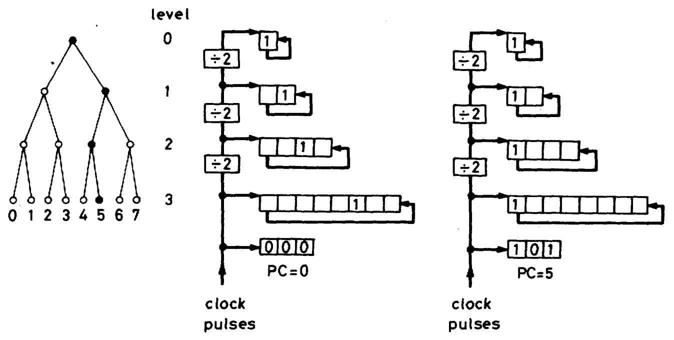

Fig. 5. Mapping a binary tree into a tree of shift registers.

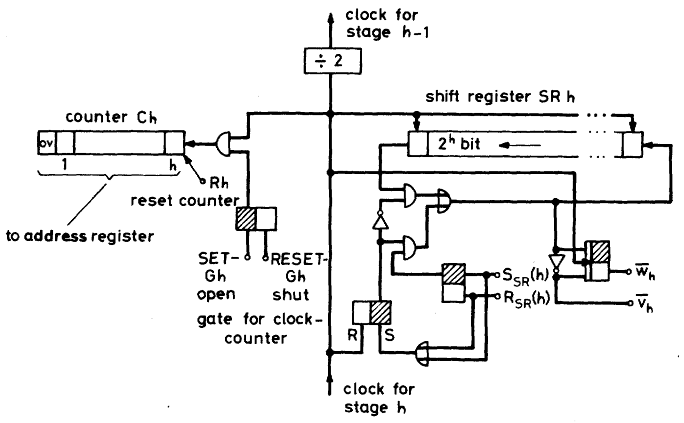

Fig. 6. A shift register with its shifting logic and a counter for the first free block.

The method of starting and stopping the index counters C_h is shown in Fig. 6, which also shows the way to get and to alter the bit values at the exits of the shift registers using $S_{SR}(h)$ and $R_{SR}(h)$. If one holds (in a separate flip-flop W_h) the bit value just shifted into the entrance of a shift register SR_h, its value W_h is the *buddy* to the exit V_h of the shift register provided the hth bit of the path selector counter is ONE, in which case an odd number of shifting pulses have been delivered to the shift register SR_h on level h. A node $T_{h,j}$ having j odd has been shifted to the exit of the shift register and the flip-flop holds the value of node $T_{h,j-1}$. Then $j = 2i + 1$ implies $T_{h,j}$ is buddy to $T_{h,j-1}$.

THE CONTROL UNIT

The control unit which realizes the algorithm is arranged around a control shift register which shifts a ONE through nine cells $S_1,\cdots,S_9$ (Fig. 8). The unit is driven by clockpulses directed either into the tree of shift registers or into the control register. Its nine exits govern the same number of different steps $C1,\cdots,C9$ of the algorithm listed below.

The allocator is set into action by an external command REQUEST which sets a flip-flop REQ, or it begins its actions when a command FREE is given to the device. The FREE command simultaneously loads into the address register

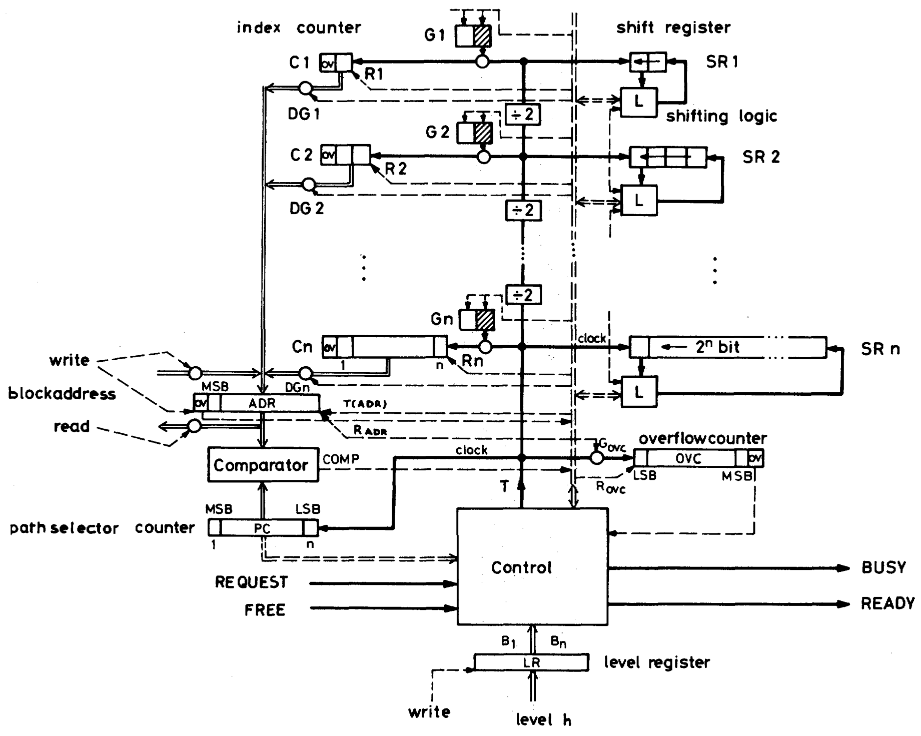

Fig. 7. Block diagram of the memory allocator.

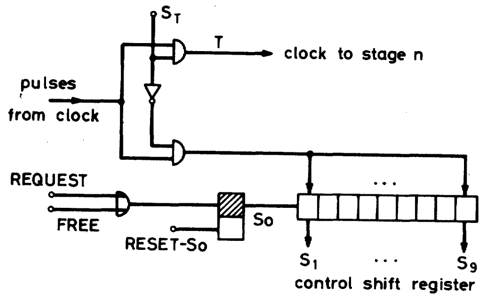

Fig. 8. Control shift register and switch for clock pulses.

the block address j_h of the block which is to be freed and sets a flip-flop FREE. With either the REQUEST or FREE command a flip-flop BUSY is set and the level number h written into a register LR activating signal line B_h there (Fig. 7).

Then clockpulses are directed into the control register and the sequence of control steps is as follows.

Step C1: (*Load index register C_h into the address register ADR*) execute in parallel for $0 \leq k \leq n$:

```
    if REQ = 1 and $B_k = 1$    then $ADR := C_k$.
```

Step C2: (*Inform the memory manager; on error stop the allocator*)

```
    if REQ = 1 and no overflow in $ADR$ then READY := 1
    else if REQ = 1 and overflow bit in $ADR$ set then
    (BUSY := 0; $S_2 := 0$; REQ := 0; ERROR := 1).
```

Step C3: (*Bring node $T_{h,j}$ to the exit of shift register SR_h*) for $PC := PC$ Step C1 until ADR do shift the tree.

Step C4: (*Mark or unmark node $T_{h,j}$*) execute in parallel for $0 \leq k \leq n$:

```
    if REQ = 1 then $S_{SR}(k) := B_k$
    else if FREE = 1 then $R_{SR}(k) := B_k$.
```

Step C5: (*Step SB and C in program* MARK *or Step B in program* UNMARK) for $i = 0$ Step C1 until $2^{n-h} - 1$ do

```
    ($PC := PC + 1$; shift the tree;
    execute in parallel for $0 \leq k \leq n$:
        if REQ = 1 then $S_{SR}(k) := 1$
        else if FREE = 1 and $V_{k-1} = 1$ then $R_{SR}(k) := 1$);
```

(the condition on i is simple to test: the $n - h$ least significant bits of PC must have become ONE).

Step C6: (*Check the buddies, Step C in program* UNMARK) for $PC := PC$ Step C1 until 2^n do

```
    (execute in parallel for $0 \leq k \leq n$:
    shift the tree;
    if FREE = 1 and $V_{k+1} = 0$ and $W_{k+1} = 0$ and $PC_h = 1$
        then $R_{SR}(k) := 1$);
```

(let PC_h denote here the bit number h in PC; see Fig. 7).

Step C7: (*Clear the index counters and the address counter, reset* READY *and clear the overflow counter OVC*)

```
    $OVC := 0$; $ADR := 0$; READY := 0;
    execute in parallel for $0 \leq k \leq n$:
        $C_k := 0$.
```

Step C8: (*Determine the address of the next free block in all levels*) for $OVC := OVC$ Step C1 until overflow in OVC do

```
    (shift the tree; execute in parallel for $0 \leq k \leq n$:
     if $V_k = 1$, then $C_k := C_k + 1$).
```

Step C9: (*Go into start position again*);

```
    BUSY := 0; REQ := 0; FREE := 0; $OVC := 0$.
```

CONCLUSION

Two full cycles of 2^n pulses each plus nine steps of the control unit will be needed until the system is ready again for a new request. Let the clock frequency be 4 MHz. Then a system managing 511 blocks, like that mentioned in the example, will output its answer to a request after 0.5 μs. It then remains busy for $2^{n+1} + 7$ clockpulses—about 130 μs.

Enlarging the system from 511 to 1023 blocks will merely increase the hardware by one further stage of shift registers (see Fig. 6) and one more bit in *ADR*, *PC*, *OVC*, and in the comparator.

The allocator might possibly be used in two types of existing situations, as follows.

1) Most minicomputers do not have means of automatic memory allocation like paging with virtual addresses. Here an allocator might be used, interfaced to the computer like any other peripheral device. Memory allocation by associative memories and virtual addressing is certainly superior but not as simple to attach to an existing minicomputer.

2) Mass storage with access times in the 100-μs range is just beginning to emerge and will need some rather fast method of allocation. Here large numbers of blocks of different sizes have to be managed and an allocator might be of help.

Preserving the internal structure of a complete binary tree in a set of shift registers and performing algorithms on the nodes by a simple control unit may find potential usage elsewhere. Patents are pending for this device.

An experimental hardware allocator handling 511 blocks is just being build for an Interdata 7/32 minicomputer.

REFERENCES

[1] D. E. Knuth, *The Art of Computer Programming*, vols. I, II. Reading, Mass.: Addison-Wesley, 1968.
[2] W. T. Comfort, "Multiword list items," *Commun. Ass. Comput. Mach.*, vol. 7, p. 357, 1964.
[3] K. C. Knowlton, "A fast storage allocator," *Commun. Ass. Comput. Mach.*, vol. 8, p. 623, 1965.
[4] S. Isoda, E. Goto, and I. Kimura, "An efficient bit table technique for dynamic storage allocation of 2^n-word blocks," *Commun. Ass. Comput. Mach.*, vol. 14, p. 589, 1971.
[5] J. W. J. Williams, "Algorithm 232: Treesort 3," *Commun. Ass. Comput. Mach.*, vol. 7, p. 347, 1964.
[6] R. W. Floyd, "Algorithm 245: Treesort 3," *Commun. Ass. Comput. Mach.*, vol. 7, p. 701, 1964.
[7] L. E. Larson and W. E. Stanton, "Algorithm for variable-length storage allocation in a circular address space," *IBM Tech. Disclosure Bull.*, vol. 15, p. 694, 1972.

Ewald von Puttkamer was born in Pommerania, Germany in 1936. He studied physics at the University of Gottingen, Gottingen, Germany and received the Diplom and Ph.D. degree from the University of Freiburg, Freiburg, Germany in 1965 and 1969, respectively. His doctoral was on coincidence measurements of photoelectrons and photoions.

Since then he has been engaged in minicomputers, interfacing experiments in atomic and molecular physics to small computers. He is now at the Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany,

Dr. von Puttkamer is a member of the Gesellschaft für Informatik.