identify all the kinds of storage which the supervisor implements. This class of channels will not be considered further.

The following simple principle is sufficient to block all legitimate and covert channels.

Masking: A program to be confined must allow its caller to determine all its inputs into legitimate and covert channels. We say that the channels are *masked* by the caller.

At first sight it seems absurd to allow the customer to determine the bill, but since the service has the right to reject the call, this scheme is an exact model of the purchase order system used for industrial procurement. Normally the vendor of the service will publish specifications from which the customer can compute the bill, and this computation might even be done automatically from an algorithmic specification by a trusted intermediary.

In the case of the covert channels one further point must be made.

Enforcement: The supervisor must ensure that a confined program's input to covert channels conforms to the caller's specifications.

This may require slowing the program down, generating spurious disk references, or whatever, but it is conceptually straightforward.

The cost of enforcement may be high. A cheaper alternative (if the customer is willing to accept some amount of leakage) is to bound the capacity of the covert channels.

Summary

From consideration of a number of examples, we have proposed a classification of the ways in which a service program can transmit information to its owner about the customer who called it. This leakage can happen through a call on a program with memory, through misuse of storage facilities provided by the supervisor, or through channels intended for other uses onto which the information is encoded. Some simple principles, which it might be feasible to implement, can be used to block these paths.

Acknowledgments. Examples 5 and 6 are due to A.G. Fraser of Bell Laboratories.

Received July 1972; revised January 1973

References

1. Lampson, B.W. Dynamic protection structures. Proc. AFIPS 1969 FJCC, Vol. 35, AFIPS Press, Montvale, N.J., pp. 27–38.
2. Schroeder, M.D., and Saltzer, J.H. A Hardware Architecture for implementing protection rings. *Comm. ACM 15*, 3 (Mar. 1972), 157–170.

Operating Systems

C. Weissman Editor

A Class of Dynamic Memory Allocation Algorithms

Daniel S. Hirschberg
Princeton University

A new dynamic memory allocation algorithm, the Fibonacci system, is introduced. This algorithm is similar to, but seems to have certain advantages over, the "buddy" system. A generalization is mentioned which includes both of these systems as special cases.

Key Words and Phrases: dynamic storage allocation, buddy system, simulation, Fibonacci, fragmentation

CR Categories: 3.89, 4.32, 4.39

Introduction

For many applications, there is a need for dynamically reserving (and releasing) variable-size blocks of contiguous memory cells. Several algorithms have been formulated and compared [3, 4]. The buddy system, introduced by Knowlton [1, 2], is preferred to other algorithms such as first-fit and best-fit on the basis of simulations conducted [3, 4].

One scheme for memory allocation transforms storage area requests (which can ask for any integral number of memory cells up to a maximum of *maxreq*) into block requests, where the number of permissible

This work was supported in part by NSF Grants GJ-965 and GJ-30126 and a National Science Foundation Graduate Fellowship.
Author's address: Department of Electrical Engineering, Princeton University, Princeton, NJ 08540.

block sizes is relatively small. Memory is originally broken into "pages" of fixed uniform size $p >$ maxreq. If a particular block size is not immediately available, a bigger block (if any are available) is broken into two smaller blocks (this is done until a block of the requested size is available). If none is available, the request will be put on a queue. Upon release, smaller blocks will be recombined with their original buddies (if they are available) to re-form bigger blocks. We are concerned here with assigning one block per area request.

In any system following this general approach (the buddy system is one such), there will be inefficiencies in memory utilization. These are caused by two factors—external fragmentation (the inability to service big requests because the available memory is contained in blocks that are of insufficient size), and internal fragmentation (the inaccessibility of unused memory that is included in blocks that are bigger than the area request that is being serviced). The inefficiency resulting from internal fragmentation has been observed to be of greater importance [5].

Expected Allocation

We derive easily computable expressions for expected waste due to internal fragmentation and expected allocation size as follows:

Let the input distribution of request sizes be described by a probability density function (pdf) and its cumulative distribution function (cdf). There are n levels of blocks of sizes $L_1, L_2, \ldots, L_n = p$. Define $L_0 = 0$, $d_i = L_i - L_{i-1}$, $i = 1 \cdots n$. Then expected waste for a single request

$$= \sum_{i=1}^{n} \int_{L_{i-1}}^{L_i} \text{pdf}(x)\,(L_i - x)\,dx,$$

$$= -\sum_{i=1}^{n} \int_{L_{i-1}}^{L_i} x\,\text{pdf}(x)\,dx + \underbrace{\sum_{1}^{n} L_i \int_{L_{i-1}}^{L_i} \text{pdf}(x)\,dx}_{J},$$

but

$$J = \sum_{1}^{n} L_i[\text{cdf}(L_i) - \text{cdf}(L_{i-1})] = p - \sum_{1}^{n} d_i\,\text{cdf}(L_{i-1})$$

thus expected waste

$$= p - \int_{0}^{p} x\,\text{pdf}(x)\,dx - \sum_{1}^{n} d_i\,\text{cdf}(L_{i-1}).$$

Expected number of cells allocated = avg request + avg waste

$$= p - \sum_{1}^{n} d_i\,\text{cdf}(L_{i-1}).$$

Example. For k levels, equally spaced from 0 to p($d = p/k$), pdf = constant = $1/p$. Expected waste

$$= p - \int_{0}^{p} x/p\,dx - p/k \sum_{1}^{k} (i-1)/k,$$

$$= p - p/2 - p(k-1)/2k = p/2k.$$

The Fibonacci System

Is there another system that follows the same general scheme as the buddy system?

If L_i and L_{i-1} are two adjacent levels, then $L_i - L_{i-1}$ must also be a level. This follows because it is possible to obtain L_{i-1} from splitting L_i. The difference, therefore, must be utilized and is a level also. Therefore the level structure in this general scheme satisfies the difference equation:

$$L_i = L_{i-1} + L_{i-k}, \qquad \text{for some } k.$$

For $k = 1$ we have the buddy system.

Following Knuth's suggestion [3], we introduce a new system, the Fibonacci system, which follows the general approach outlined above. It is similar to the buddy system except, instead of breaking a block of size 2^n into two blocks of size 2^{n-1}, it breaks a block of size f_n (the nth Fibonacci number) into blocks of sizes f_{n-1} and f_{n-2}. In other words, the Fibonacci system corresponds to using $k = 2$ in the scheme above. Other possibilities which are left for future research are those where $k = 3, 4, \ldots$.

A Simulation Experiment

A simulation was conducted comparing the buddy and Fibonacci systems. The input distribution of request sizes was similar to the buffer request distribution of the Univac 1108 Exec 8 system at the University of Maryland [4]. This distribution is indicated in Appendix A. As in [4], input request arrival times were Poisson distributed with exponential hold times (defined to be the length of time a block is in use). Output included snapshots of the system giving number of blocks allocated so far, presently being serviced, and on the queue, percentage utilization of memory, size distribution of free blocks, and size distribution of block requests that are on the queue. When blocks become available, the biggest block request on the queue (needing that size or less) is fulfilled. After a predetermined number of requests had been made, the system was allowed to wind down and a final output for the run included the following: total simulation time, average queue length, average nonzero queue length (the averages were taken from time t (equal to average hold time) after start-up to avoid initial transients until initiation of wind-down to avoid final transients), size distribution of blocks which had been queued, and the time spent on the queue, as well as

Fig. 1.

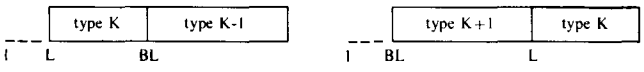
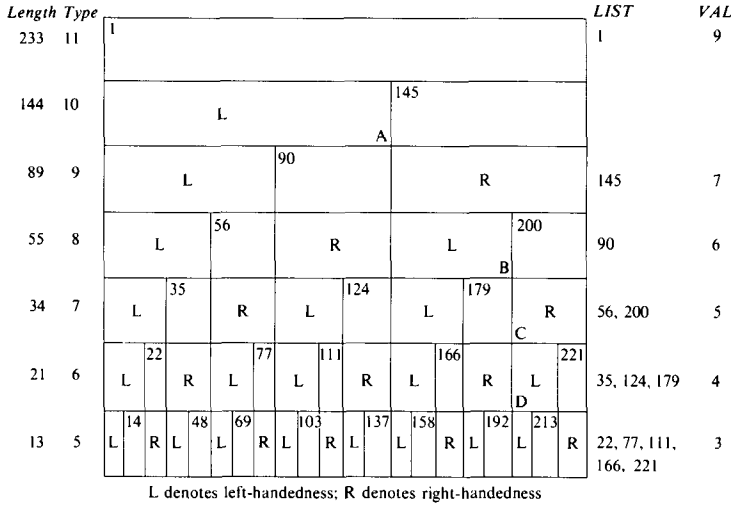


Fig. 2.



L denotes left-handedness; R denotes right-handedness

total memory requested, allocated, and allocation/request ratio. The allocation/request ratio indicates the percentage of unused memory that is tied up in blocks and hence inaccessible. Thus, this ratio is a measure of internal fragmentation.

Several runs were made with each of several average hold times (which varied the loading factor). Each run using the Fibonacci system was also run with identical inputs using the buddy system.

Results of Simulation

The results closely agreed with predictions for internal fragmentation for both systems:

Allocations/requests		*buddy*	*Fibonacci*
expected ratio		1.381	1.250 (see App. A)
actual ratio	max	1.388	1.252
	avg	1.381	1.248
	min	1.365	1.236

The expected ratios are derived (assuming the pdf in [4]) in Appendix A. Note that 1.250 means that one-fourth more space is allocated than requested. The actual ratio figures are the results of the simulations conducted. The average actual ratio closely agrees with the expected ratio. The variance is not excessively large.

Several other items were noted (comparisons of request sizes and times were made for the same inputs).

1. Total simulation times were roughly the same; however, the average queue length, number of requests queued, and total time spent on queue were consistently higher for buddy than for Fibonacci (at times, a factor of 2 or more).
2. Both systems were able to service big requests under moderate load conditions (queue not growing indefinitely) indicating that recombinations were occurring fairly often. Under saturation conditions, however, both systems demonstrated a build-up of predominantly big requests on the queue even though big requests on the queue had priority over smaller ones (to minimize the aforementioned build-up).
3. It was observed that, under moderate load conditions, buddy has 95 percent utilization of memory while Fibonacci has only 90 percent, but buddy's 95 is servicing less requested space than Fibonacci's 90 percent.

Overall, the Fibonacci system seems preferable to the buddy system.

Received September 1972; revised February 1973

Appendix A. Evaluation of Expected Allocation/Request Ratio

S_i	del	cdf (%)	pdf (%)	contr to avg req
2		0		
8	6	36	6	1.98
10	2	44	4	.76
15	5	54	2	1.30
25	10	84	3	6.15
30	5	94	2	2.8
35	5	96.5	.5	.825
40	5	97.5	.2	.38
50	10	98.5	.1	.455
70	20	99.3	.04	.484
100	30	99.6	.01	.2565
200	100	100	.004	.602

Average request = 15.9925
del = $S_i - S_{i-1}$.
pdf_i applies to range $[S_{i-1}, S_i]$.
$cdf_i = cdf_{i-1} + del \times pdf_i$.
contr to avg req = $\sum_{j=l_{i-1}+1}^{l_i} j \cdot pdf_i$.
The pdf corresponds to that in [4].

Buddy			*Fibonacci*			
$L_i = d_{i+1}$	cdf_i (%)	$d_{i+1} \times cdf_i$	L_i	d_{i+1}	cdf_i (%)	$\times$
4	12	48	3	2	6	12
8	36	288	5	3	18	54
16	57	912	8	5	36	180
32	95	3040	13	8	50	400
64	99.06	6340	21	13	72	936
128	99.712	12762	34	21	96	2016
			55	34	98.7	3356
			89	55	99.49	5472
			144	89	99.776	8880
	sum =	23390			sum =	21306

Avg alloc = $p - \sum_1^n d_i \, cdf\,(L_{i-1})$.
Avg alloc: 256 − 233.9 = 22.1 233 − 213 = 20
Alloc/req ratio: 22.1/16 = 1.381 20/16 = 1.250

Appendix B. Algorithm A and Algorithm R

Blocks are available in ten sizes. They are referred to (from the smallest through largest) as types 2 through 11 (sizes 3 through 233).

Algorithm A. Block Allocation for the Fibonacci System

INPUT: K = type wanted

OUTPUT: L = starting location, K = type allocated; if not allocatable at present, $L = 0$

Address space starts at location 1

$AVAIL(i)$ = location of a free block of type i
= 0 if none available

$LINK(L)$ = location of next free block of same type as block starting at location L
= 0 if no more available

$TYPE(L)$ = type of block (2 through 11) that starts at location L

$FIBSIZ(i)$ = size of block of type i

1. Find min $i \geq K$ such that $AVAIL(i) \neq 0$ (if none, $L := 0$ **return**)
2. Remove block from list: $L := AVAIL(i)$;
   $AVAIL(i) := LINK(L)$
3. **if** $i = K$ **then return**
4. Split block: **if** $i = 3$ **then** [$K := i$; **return**]; $i := i - 2$;
   $P := L + FIBSIZ(i+1)$; $TYPE(P) := i$; $TYPE(L) := i + 1$
5. **if** $i + 1 = K$ **then goto** 7
6. Put bigger buddy on available list:
   $LINK(L) := AVAIL(i+1)$; $AVAIL(i+1) := L$; $L := P$; **goto** 3
7. Put smaller buddy on available list and return:
   $LINK(P) := AVAIL(i)$; $AVAIL(i) := P$; **return**

Algorithm R. Block Release for the Fibonacci System

INPUT: block starts at location L and is of type K ($TYPE(L) = K$)

1. **if** $K = MAXTYP$ **then goto** 8 [$MAXTYP = 11$]
2. Is buddy smaller or bigger? [See Appendix C]
   $N := (L \bmod 233) - FIBSIZ(K+1)$;
   if $K = 2$ **then goto** 4;
   for $i := 1$ **to** 13 **do**
   [**if** $LIST(i) > N$ **then goto** 3;
   if $LIST(i) = N$ **then**
   [**if** $K \leq VAL(i)$ **then goto** 4 **else goto** 3]]
3. Smaller buddy at higher location number
   (BL = buddy loc.; BK = buddy type):
   $BL := L + FIBSIZ(K+1)$; $BK := K - 1$; **goto** 5
4. Bigger buddy at lower location number:
   $BL := L - FIBSIZ(K+1)$; $BK := K + 1$
5. **if** block at buddy location is wrong size or not available **then goto** 8;
   Remove buddy from list and combine:
   if $AVAIL(BK) = BL$ **then** [$AVAIL(BK) := LINK(BL)$; **goto** 7];
   $JJ := AVAIL(BK)$
6. $j := JJ$; $JJ := LINK(j)$;
   if $JJ \neq BL$ **then goto** 6;
   $LINK(j) := LINK(BL)$
7. **if** $BL < L$ **then** $L := BL$;
   $K := max(K,BK) + 1$;
   goto 1
8. Put block on list K:
   $LINK(L) := AVAIL(K)$; $AVAIL(K) := L$; $TYPE(L) := K$;
   return

$FIBSIZ(2{:}MAXTYP)$:
3, 5, 8, 13, 21, 34, 55, 89, 144, 233

$LIST(1{:}13)$:
1 22 35 56 77 90 111 124 145 166 179 200 221

$VAL(1{:}13)$:
9 3 4 5 3 6 3 4 7 3 4 5 3

Appendix C. Buddy Finding in the Fibonacci System

We are given a block of type K starting at location L'. Since memory is broken into pages of size 233, we find the location relative to the beginning of the page $L = (L' \bmod 233)$. We wish to find the type BK and location BL of the original buddy of this block. In splitting blocks we always put the bigger buddy at the lower location number. Therefore, if $BK = K + 1$, then $BL = L - FIBSIZ(K+1)$; if $BK = K - 1$, then $BL = L + FIBSIZ(K)$ (see Figure 1).

Method 1 (used in Algorithm R). We first determine the Fibonacci storage layout (see Figure 2). From this we construct $LIST$, which is simply an ordered sequence of locations (mod 233) at which a block of type 5 or higher can start. VAL is constructed to correspond to $LIST$ such that blocks of type $VAL_i + 2$ or lower start at location $LIST_i$.

We wish to see if it is possible to have a bigger buddy (at lower location number). That is, can a block of type $BK = K + 1$ start at location $BL = L - FIBSIZ(K+1)$ and be "left-handed"? We note that left-handedness of a block of type T at location L occurs if and only if a block of type $T + 1$ can start at location L. Therefore we test to see if a block of type $BK + 1$[$=K+2$] can start at location BL [$=L-FIBSIZ(K+1)$] which happens if and only if BL is on $LIST$ and the corresponding VAL is at most $(BK+1)-2$ [$=K$].

Method 2. If $L = 1$, then the block must be left-handed, and therefore the buddy is smaller at a higher location, and $BK = K - 1$, $BL = L + FIBSIZ(K)$.

If $L > 1$, then build up to L by successively adding the biggest $FIBSIZ$ possible without exceeding L.

Then $L - 1 = FIBSIZ(i_1) + \cdots + FIBSIZ(i_m)$, where $i_1 > \cdots > i_m > K$.

If $i_m > K + 1$, then buddy is right-handed, $BK = K - 1$.

Otherwise $i_m = K + 1$, and buddy is left-handed, $BK = K + 1$.

As an example (see Figure 2) we add sizes A and B to get C or D. $TYPE(A) = 10$, $TYPE(B) = 8$, $TYPE(C) = 7$, $TYPE(D) = 6$. Therefore C's buddy must be left-handed ($8 = 7 + 1$); D's buddy must be right-handed ($8 = 6 + 2$).

References

1. Knowlton, K.C. A fast storage allocator, *Comm. ACM 8*, 10 (Oct. 1965), 623–625.
2. Knowlton, K.C. A programmer's description of *L6*. *Comm. ACM 9*, 8 (Aug. 1966), 616–625.
3. Knuth, D.E. *The Art of Computer Programming, Vol. 1* (2nd printing). Addison-Wesley, Reading, Mass., 1968, pp. 435–455.
4. Minker, J., et al. Analysis of data processing systems. Tech. Rept. 69–99, U. of Maryland, College Park, Md., 1969.
5. Purdom, P., and Stigler, S. Statistical properties of the buddy system. *J. ACM 17*, 4 (Oct. 1970), 683–697.