di Sergio Polini

Exec senza limiti

«You certainly have my permission to write about and distribute EXECSWAP to your readers». Con queste parole Kim Kokkonen, presidente della Turbo Power Software, mi ha cortesemente autorizzato a mettervi a parte di una efficace tecnica da lui messa a punto per mantenere i benefici della procedura Exec superandone i limiti. La tecnica era stata illustrata nel numero di aprile del Dr. Dobb's Journal *e credo dobbiamo tutti ringraziare Kokkonen di aver consentito la pubblicazione del suo codice su MC*

La Turbo Power Softare offre diversi prodotti per aiutare il programmatore Turbo Pascal a realizzare applicazioni «serie». Il *B-Tree Filer* permette la gestione di basi di dati in modo compatibile con il *Database Toolbox* della Borland anche per reti locali o in genere per la multiutenza, il *Turbo Professional* comprende librerie di funzioni per programmi residenti, per l'aritmetica BCD, per la creazione di maschere per l'input di dati e di sistemi di help sensibile al contesto (con tanto di uso del mouse), per la manipolazione di stringhe con più di 255 caratteri o di array oltre i 64K, per l'accesso alla memoria EMS, ecc. Il *Turbo Analyst* consente di analizzare la struttura di un programma e di valutarne l'efficienza con ben tre *profiler*: con uno si può verificare quanto del tempo totale di esecuzione è stato consumato in ogni funzione o procedura, con un altro si può verificare quante volte è stata eseguita ogni linea del sorgente (utilissimo per il test di un programma: una linea eseguita zero volte è una linea non testata, quindi possibile sede di bug!), con un terzo si possono ottimizzare le routine in linguaggio macchina.

Non è una descrizione completa; ora ci interessa soprattutto un «pezzo» del *Turbo Professional* (del quale è stata già annunciata una nuova versione «object oriented» per il TP 5.5): la funzione *ExecWithSwap*, che si propone come alternativa alla procedura *Exec* della Borland in quanto non solo ne elimina gli inconvenienti, ma ne amplia le possibilità di utilizzo. Prima di vedere la soluzione, però, sarà meglio capire il problema.

La gestione della memoria nel DOS

Quando accendiamo il PC, il BIOS va a vedere quanta memoria è fisicamente disponibile e ne scrive l'ammontare, in Kbyte, in una zona della sua area dati (all'indirizzo $0040:$0013; l'informazione può essere ottenuta mediante l'interrupt $12, che ritorna quel numero nel registro AX).

Il vecchio DOS 1.x non se ne curava granché: il COMMAND.COM si limitava a prenderne nota in una sua locazione di memoria, ma per il resto ogni programma che veniva eseguito si allocava per sé tutta la RAM disponibile, senza alcun intervento del DOS.

A partire dal DOS 2.0 le cose sono cambiate sostanzialmente. Da allora il DOS gestisce la memoria mediante una lista di *memory control blocks*. Ogni «blocco» è lungo 16 byte (un paragrafo) e rappresenta in pratica l'intestazione di un'area di memoria, che può essere già allocata a beneficio di un programma oppure libera e disponibile. Non tutti quei 16 byte vengono utilizzati: nel primo c'è una 'M' se l'area controllata dal blocco è allocata o una 'Z' se è libera; il secondo e il terzo sono nulli se l'area è libera, altrimenti contengono l'indirizzo del *Program Segment Prefix* (PSP) del programma che la occupa; il quarto e il quinto contengono la dimensione dell'area e consentono quindi di determinare l'indirizzo del blocco successivo. In ogni area di memoria «occupata», subito dopo i 16 byte del blocco abbiamo il PSP del programma, quindi il programma stesso.

Vi sono tre funzioni del DOS che intervengono su questa struttura: la $48 alloca un blocco di memoria, la $49 lo rilascia, la $4A modifica l'ampiezza del blocco il cui segmento sia contenuto nel registro ES. Per vedere alcuni possibili usi di queste funzioni, occorre esaminare cosa succede quando viene eseguito un programma.

Se si tratta di un programma con estensione COM, il DOS gli alloca tutta la RAM disponibile; sarà quindi compito dello stesso programma verificare in primo luogo se la memoria di cui dispone gli è sufficiente e poi, in caso affermativo, eventualmente liberare quella in eccesso: si può usare per questo la funzione $4A. Chi programma in assembler sa che il codice di ogni file COM deve iniziare all'indirizzo $100 (256 in decimale), in quanto nei primi 256 byte il DOS pone il *Program Segment Prefix*. Magari ricorderà anche che, quando il programma parte, tutti i registri di segmento (CS, DS, ES, SS) contengono l'indirizzo del PSP; ciò permette di chiamare la $4A solo indicando nel registro BX l'ammontare in paragrafi della memoria che occorre. In questo modo il programma può restituire al DOS quella che non gli serve.

Se si tratta di un programma EXE le cose sono un po' più complicate; per ora diciamo solo che ogni file EXE ha uno *header* in cui sono contenute nu-

merose informazioni, tra cui l'occupazione di memoria richiesta dal programma per codice, dati, stack e heap. Ciò consente al DOS di lasciare libera la memoria non necessaria.

Se un file EXE non richiede espressamente tutta la RAM disponibile (o se poi la restituisce al DOS con la funzione $4A, come i file COM), può in un secondo tempo usare le funzioni $48 e $49 per gestirsi il suo heap in aree di memoria poste «sopra» quella in cui sono codice, dati e stack del programma.

La gestione della memoria nel Turbo Pascal

I file EXE generati dal Turbo Pascal si comportano però in altro modo, in particolare per quanto riguarda lo heap: questo è sempre collocato nella stessa area di memoria in cui risiedono anche codice dati e stack. Non è una differenza da poco. Vediamo un po' più in dettaglio cosa avviene.

Supponiamo di compilare il programma MODEXETP.PAS (figura 1, su cui torneremo tra breve) dopo aver scelto l'opzione «Segments» nel menu «Options/Linker/Map file». Oltre al file EXE, viene così generato anche un file MODEXETP.MAP, riprodotto nella figura 2. Qui possiamo vedere come sono disposti i diversi segmenti del programma: abbiamo all'inizio (subito dopo il *Program Segment Prefix*) il segmento MODEXETP, contenente il codice prodotto dalla compilazione del nostro sorgente; seguono poi analoghi segmenti per le unit usate esplicitamente (CRT e DOS) o implicitamente (SYSTEM). Più sopra (ad indirizzi di memoria più alti) troviamo il segmento dedicato ai dati ed uno — di lunghezza nulla — che costituisce la base dello heap nei programmi che usano le procedure *New* o *GetMem*.

Dobbiamo sottolineare due cose. Innanzitutto notiamo che la base dello heap è parte integrante dell'organizzazione del programma in segmenti: ciò

```
program ModExeTP;
uses Crt, Dos;
const
  P16 : longint = 16;          (* Byte in un paragrafo *)
type
  HeaderFmt = record
    Sigla       : word;        (* 'MZ' (sigla di Mark Zbikowski) *)
    Resto       : word;        (* (Lunghezza del file senza header) mod 512 *)
    TotLen      : word;        (* (Lunghezza del file) div 512 *)
    NumReloc    : word;        (* Numero elementi da rilocare *)
    HeadLen     : word;        (* Lunghezza dell'header in paragrafi *)
    ParaMin     : word;        (* Numeri minimo e massimo di paragrafi *)
    ParaMax     : word;        (*   da allocare per heap e stack *)
    InitSS      : word;        (* Distanza in paragrafi di SS dall'inizio *)
    InitSP      : word;        (* Valore iniziale di SP *)
    CheckSum    : word;
    InitIP      : word;        (* Valore iniziale di IP *)
    InitCS      : word;        (* Distanza in paragrafi di CS dall'inizio *)
    PrimoReloc  : word;        (* Indirizzo del primo elemento da rilocare *)
    NumOverlay  : word;        (* Numero di overlay *)
  end;
var
  Header        : HeaderFmt;
  NomeFile      : PathStr;
  FileExe       : file of HeaderFmt;
  i             : integer;
  Param         : string;
  StackLen      : integer;      (* in KByte *)
  ExtraHeap     : integer;      (* diff. tra MaxHeap e MinHeap in KByte *)
  OrigStackLen  : word;         (* in paragrafi *)
  NumPara       : word;
procedure Uso;
begin
  Writeln('Uso: modexetp ˇ-sh <numero di Kbyte>é file');
  Halt(1);
end;
procedure Assegna(Stringa: PathStr; var Valore: integer);
var
  Errori: integer;
begin
  Val(Stringa, Valore, Errori);
  if (Errori <> 0) or (Valore < 0) then Uso;
end;
procedure PrintHeader;
begin
  Writeln('Memoria minima per stack e heap','':19,Header.ParaMin * P16);
  Writeln('Memoria massima per stack e heap','':18,Header.ParaMax * P16);
  Writeln('.. di cui dedicata allo stack','':21,Header.InitSP);
  Writeln('Differenza tra MaxHeap e MinHeap','':18,
    (Header.ParaMax - Header.ParaMin) * P16);
  Writeln;
end;
begin
  CheckBreak := FALSE;
  if ParamCount < 1 then Uso;
  i := 1;
  StackLen := -1; ExtraHeap := -1;
  repeat
    Param := ParamStr(i);

  if Paramˇ1é = '-' then begin
    case Paramˇ2é of
      'h': begin Inc(i); Assegna(ParamStr(i), ExtraHeap); end;
      's': begin Inc(i); Assegna(ParamStr(i), StackLen); end;
      else Uso
    end
  end
  else begin
    NomeFile := Param;
    if Pos('.',NomeFile) = 0 then NomeFile := NomeFile + '.exe'
  end;
  Inc(i)
until i > ParamCount;
Assign(FileExe, NomeFile);
(*$I-*) Reset(FileExe); (*$I+*)
if IOResult <> 0 then begin
  Writeln('Errore apertura ', NomeFile);
  Halt(2);
end;
Read(FileExe, Header);
if (ExtraHeap >= 0) or (StackLen >= 0) then begin
  PrintHeader;
  if ExtraHeap >= 0 then
    Header.ParaMax := Header.ParaMin + ExtraHeap * 64;
  if StackLen >= 0 then begin
    OrigStackLen := Header.InitSP div 16;
    if Header.InitSP mod 16 <> 0 then Inc(OrigStackLen);
    Header.InitSP := StackLen * 1024;
    Header.ParaMin := Header.Paramin - OrigStackLen + StackLen * 64;
    Header.ParaMax := Header.Paramax - OrigStackLen + StackLen * 64;
  end;
  Seek(FileExe, 0);
  Write(FileExe, Header);
end;
PrintHeader;
Close(FileExe)
end.
```

Figura 1 - Un equivalente di EXEMOD per i file EXE generati dal Turbo Pascal. Una curiosità: la catena dei blocchi di memoria gestiti dal DOS è governata dalle lettere M e Z (come spiegato nel testo); i primi due byte di ogni file EXE sono $4D e $5A, ancora M e Z. Non può essere un caso che il principale responsabile dello sviluppo del DOS 2.0 si chiamasse Mark Zbikowski.

```
Start   Stop    Length Name                 Class

00000H 004C2H 004C3H MODEXETP               CODE
004D0H 004D0H 00000H DOS                    CODE
004D0H 00AE2H 00613H CRT                    CODE
00AF0H 01876H 00D87H SYSTEM                 CODE
01880H 01D13H 00494H DATA                   DATA
01D20H 05D1FH 04000H STACK                  STACK
05D20H 05D20H 00000H HEAP                   HEAP
```

Figura 2 - Il file MODEXETP.MAP, generato dal Turbo Pascal se compiliamo MODEXETP.PAS dopo aver scelto l'opzione «Segments» nel menu «Options/Linker/Map file».

perché, come dicevamo, i programmi compilati con il Turbo Pascal non chiedono al DOS la memoria per lo heap dopo l'avvio, ma al momento stesso del caricamento in memoria. In altri termini, un programma Turbo Pascal deve sapere a priori quanta memoria destinare allo heap, in quanto questa informazione deve essere contenuta nello header del file EXE. In esso vi sono due campi (chiamati *ParaMin* e *ParaMax* nel listato in figura 1) che devono contenere l'ammontare minimo e massimo della memoria necessaria per lo stack e, appunto, per lo heap. Il mese scorso abbiamo ricordato la direttiva $M e i suoi tre argomenti: dimensione dello stack e dimensioni minima e massima dello heap, con valori di default, rispettivamente, di 16384, 0 e 655360 byte; *ParaMin* conterrà la somma della dimensione dello stack e di quella minima dello heap (più altri spiccioli che ora non ci interessano), *ParaMax* la somma della dimensione dello stack e di quella massima dello heap (più gli stessi spiccioli). Ciò consente appunto al DOS di attribuire subito al programma tutta la memoria che gli serve, heap compreso, e consente al programma di gestire la memoria dinamica senza ulteriori chiamate al DOS, quindi con notevole efficienza. Un primo, solo apparente, aspetto negativo è rappresentato dal fatto che è praticamente impossibile prevedere quanta RAM sarà disponibile sulla macchina su cui il programma dovrà girare, ma a questo si rimedia facilmente chiedendo «tutto il mazzo»: perché il programma possa essere caricato dal DOS è sufficiente infatti che nella memoria disponibile entrino codice dati e *ParaMin*; se non c'è spazio per *ParaMax*, la conseguenza è solo che al programma viene data tutta la RAM disponibile, quale ne sia l'ammontare. Ecco perché i valori di default sono 0 per la dimensione minima dello heap e 655360 (cioè 640K) per quella massima. Il secondo, reale, aspetto negativo è rappresentato dal fatto che questa strategia porta il programma ad appropriarsi sempre di tutta la RAM disponibile, salvo intervenire con la direttiva $M.

Qui il problema è reale perché, come abbiamo visto la volta scorsa, la procedura *Exec* ha bisogno di trovare un'area di memoria non occupata dal programma per poterne eseguire un altro. Dovremmo quindi dare valori appropriati alla direttiva $M, ma per far questo dovremmo conoscere esattamente la quantità di memoria disponibile prima della esecuzione del programma: può andar bene se usiamo il programma solo sulla nostra macchina, non certo per un programma destinato a girare su altre. Una prima soluzione potrebbe essere appunto MODEXETP: analogo all'EXEMOD fornito dalla Microsoft per alcuni suoi compilatori, consente di cambiare la dimensione dello stack e la differenza tra dimensioni minima e massima dello heap di un programma senza ricompilarlo; non fa altro che modificare l'*header* di un file EXE (ne sconsiglio l'uso per programmi non compilati con il Turbo Pascal). Potete provarlo su se stesso:

```
modexetp modexetp
```

mostra i valori di *ParaMin* e *ParaMax*, la dimensione dello stack e la differenza tra *MaxHeap* e *MinHeap*;

```
modexetp -s 2 -h 64 modexetp
```

assegna due Kbyte allo stack e rende *MaxHeap* uguale a *MinHeap* più 64K. La soluzione di Kim Kokkonen è molto più efficace; in realtà vi propongo MODEXETP solo per consentirvi di «toccare con mano» quei meccanismi che stanno dietro la direttiva $M.

L'altra cosa da sottolineare — e da tenere bene a mente per usare al meglio la soluzione proposta da Kokkonen — è che i segmenti dedicati alle unit sono disposti in ordine inverso: all'indirizzo più basso, subito dopo il codice del sorgente principale, troviamo *la unit elencata per ultima nella istruzione* **uses**; all'indirizzo più alto, subito prima del segmento dei dati, la unit System.

ExecSwap

Avrete sicuramente già indovinato. Si tratta «semplicemente» di liberare memoria scaricando su un file di *swap* parte di quella occupata dal programma, eseguire poi un altro programma come se usassimo la procedura *Exec*, ripristinare infine il contenuto della memoria. In verità, tuttavia, c'è dell'altro: ad esempio si può usare la memoria EMS invece che un file di swap e, con un minimo di attenzione, si riesce quasi a far sparire dalla memoria il programma che usa *ExecWithSwap*, con il risultato che è possibile «exeguire» anche programmi decisamente ingombranti.

La unit EXECSWAP.PAS (figura 3; per motivi di spazio, l'implementazione della funzione *InitExecSwap* verrà pubblicata il mese prossimo) propone una **interface** con le due funzioni *InitExecSwap* e *ExecWithSwap* e la procedura *ShutdownExecSwap*. L'uso di *ExecWithSwap* è praticamente identico a quello della procedura *Exec*, compresa la raccomandazione di chiamare *SwapVectors* subito prima e subito dopo, come avevamo visto il mese scorso; la sola differenza è che, trattandosi di una funzione, ritorna un valore (un codice d'errore). Quanto a *ShutdownExecSwap*, si tratta solo di ... ricordarsi di chiamarla dopo *ExecWithSwap*, tranne nel caso che il programma termini; il suo unico compito è di rilasciare la memoria EMS utilizzata o di cancellare il file di *swap*, secondo il caso.

I due parametri di *InitExecSwap* richiedono qualche parola in più. L'inizio dell'area di memoria da scaricare tem-

```
(*$S-*)
unit ExecSwap;

interface

const
  BytesSwapped : longint = 0;      (* byte 'swappati' su EMS o disco *)
  EmsAllocated : boolean = FALSE; (* TRUE se si usa l'EMS *)
  FileAllocated: boolean = FALSE; (* TRUE se si usa un file *)

function ExecWithSwap(Path, CmdLine: string): word;
  (* Exec con swap su disco o EMS *)
function InitExecSwap(LastToSave: pointer; SwapFileName: string): boolean;
  (* Ritorna TRUE se l'inizializzazione e' andata a buon fine *)
procedure ShutdownExecSwap;
  (* 'Dealloca' l'area di swap *)

implementation

var
  EmsHandle : word;
  FrameSeg  : word;
  FileHandle: word;
  SwapName  : string°80é;
  SaveExit  : pointer;

(*$L EXECSWAP *)
function ExecWithSwap(Path, CmdLine: string): word; external;
procedure FirstToSave; external;
function AllocateSwapFile: boolean; external;
procedure DeallocateSwapFile; external;

(*$F+*)
(* Le seguenti routine potrebbero anche essere elencate nella interface *)
function EmsInstalled: boolean; external;
function EmsPageFrame: word; external;
function AllocateEmsPages(NumPages: word): word; external;
procedure DeallocateEmsHandle(Handle: word); external;
function DefaultDrive: char; external;
function DiskFree(Drive: byte): longint; external;

procedure ExecSwapExit;
begin
  ExitProc := SaveExit;
  ShutDownExecSwap
end;
(*$F-*)

procedure ShutdownExecSwap;
begin
  if EmsAllocated then begin
    DeallocateEmsHandle(EmsHandle);
    EmsAllocated := FALSE
  end else if FileAllocated then begin
    DeallocateSwapFile;
    FileAllocated := FALSE

  end
end;


function PtrDiff(H, L: Pointer): longint;
type
  OS = record O, S: word end; (* per forzare una conversione di tipo *)
begin
  PtrDiff := (longint(OS(H).S) shl 4 + OS(H).O) -
             (longint(OS(L).S) shl 4 + OS(L).O)
end;

function InitExecSwap(LastToSave: pointer; SwapFileName: string): boolean;
 (* Verra' pubblicata il mese prossimo (lo spazio e' tiranno!) *)
end;

end.
```

Figura 3 - Il sorgente della unit EXECSWAP. Sia la funzione InitExecSwap che quelle «external» (contenute in EXECSWAP.ASM) verranno illustrate e commentate nei prossimi numeri.

poraneamente su EMS o su disco sarà ovviamente subito dopo il codice della funzione *ExecWithSwap*, in quanto non ci si può sbarazzare del codice che dovrà poi ripristinare il contenuto originario della memoria. Questo comporta che la parte del programma che non verrà «swappata» sarà quella compresa tra il PSP e la funzione *ExecWithSwap* più la funzione stessa; ne segue che sarà opportuno indicare la unit EXECSWAP per ultima nella istruzione **uses**, e che la chiamata avverrà preferibilmente mediante istruzioni poste nel sorgente del file principale, quello con intestazione **program**, il quale sarà il più breve possibile. Per evitare eccessive acrobazie (può infatti risultare necessario chiamare *ExecWithSwap* da più parti, anche da altre unit), Kokkonen consiglia di dichiarare in una unit da elencare tra le prime alcune variabili di tipo procedura, alle quali il codice del file principale assegnerà gli indirizzi delle routine di EXECSWAP; sarà quindi possibile ad ogni unit elencata dopo quella chiamare le routine mediante tali variabili.

La fine dell'area da «swappare» va invece passata nel primo parametro di *InitExecSwap*. Ci sono tre possibilità. Se il programma non fa uso di memoria dinamica si può usare la variabile predefinita *HeapOrg*, nella quale è contenuto l'indirizzo della base dello heap; in questo modo si potranno poi ripristinare codice dati e stack. Se il programma usa variabili dinamiche, ma non le libera se non alla fine, si deve scaricare su disco anche lo heap ma si può trascurare la *free list* (la lista dei blocchi di memoria deallocati): si può cioè usare la variabile *HeapPtr*, che contiene l'indirizzo più alto dello heap. Se infine si deve salvare anche la *free list*, si deve usare l'espressione:

```
Ptr( Seg(FreePtr^) + $1000, 0 )
```

per la quale vi rimando all'esauriente manuale del compilatore.

Nelle prossime due puntate vedremo in dettaglio come funziona il tutto. Potete intanto trovare il sorgente completo di EXECSWAP e di MODEXETP su MC-Link. A presto.

OK,KAO!

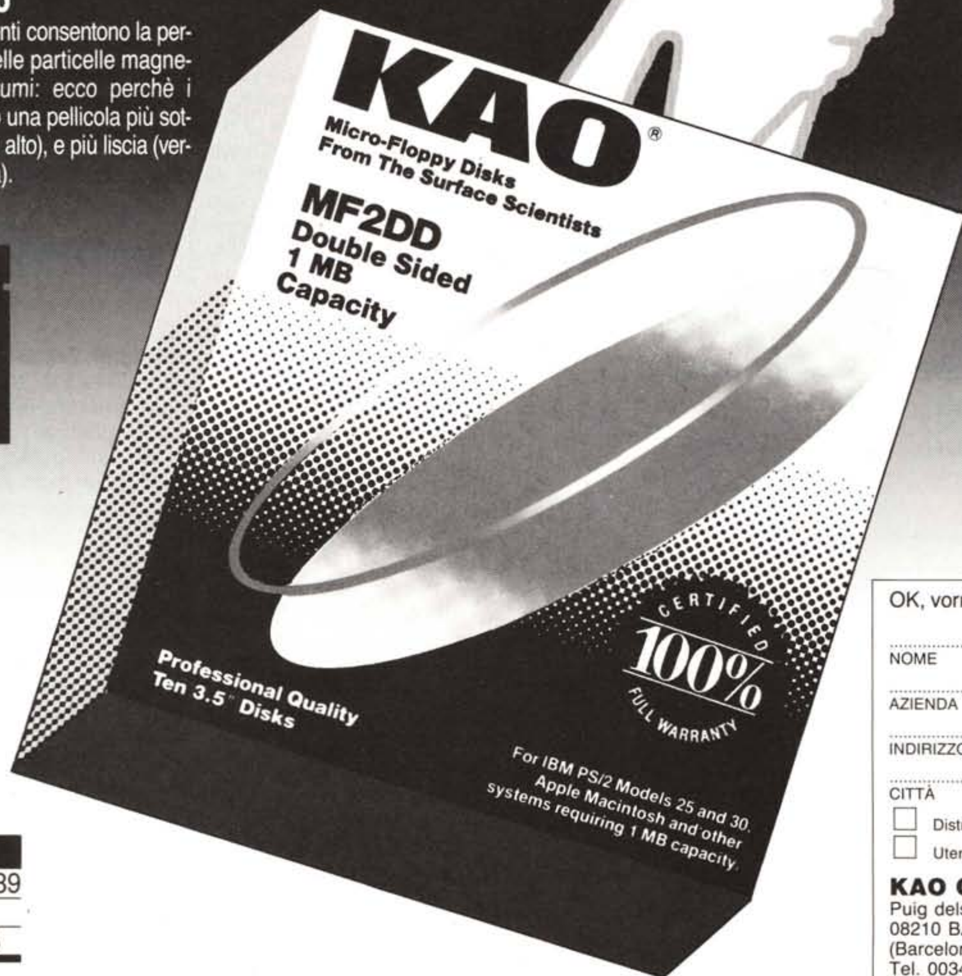
LA RICERCA KAO
Quando la KAO vanta la qualità dei propri dischetti sa quello che dice: infatti i dischetti vengono prodotti dalla KAO stessa. E la KAO ha 95 anni di esperienza nella tecnologia dei fenomeni di superficie (emulsioni, dispersioni, ecc.) e nella ricerca e sviluppo dei prodotti chimici per l'industria.

L'ESPERIENZA KAO
La KAO fornisce materie prime, additivi e dischetti diffusi in tutto il mondo: è da questa conoscenza a fondo del dischetto, dalle materie prime al prodotto finito, che sboccia oggi il dischetto firmato KAO.

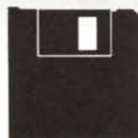
I DISCHETTI KAO
Speciali polidisperdenti consentono la perfetta separazione delle particelle magnetiche evitando i grumi: ecco perchè i dischetti KAO hanno una pellicola più sottile (clipping level più alto), e più liscia (verniciatura omogenea).

Un polimero di poliuretano ad alta coesione crea una pellicola con elevatissima resistenza all'usura: e questo, unitamente allo speciale lubrificante incorporato nella pellicola riduce al minimo l'attrito della testina col dischetto. Come dire: niente usura, niente polvere.

LA GAMMA KAO
Con prezzi particolarmente interessanti, a faccia singola oppure doppia, di doppia o alta densità, da 3½'' o da 5¼'', la gamma dei dischetti KAO è completa, la qualità indiscussa, la garanzia totale.
Perfino in condizioni estreme di temperatura e umidità i dischetti KAO resistono a oltre 20 milioni di cicli di scrittura-lettura.



5¼'' MD-2D MD-2HD



3½'' MF-2DD MF-2HD

SMAU
5-9 OTTOBRE 89
PAD. 15
STAND H 08

OK, vorrei maggiori informazioni;

NOME

AZIENDA

INDIRIZZO

CITTÀ PROV.

☐ Distributore ☐ Rivenditore
☐ Utente finale ☐ Duplicatore

KAO CORPORATION SA
Puig dels Tudons 10
08210 BARBERÀ DEL VALLÈS
(Barcelona) ESPAÑA
Tel. 00343.718.23.13
Fax 00343.718.98.29

MC

KAO®

DOMINO

Ecco perchè in tutto il mondo i computers dicono OK,KAO!

DISTRIBUTORI: TECNOCENTRO Via Mac Mahon, 50 - 20155 MILANO - Tel. (02) 3492063 • **TPA** Via Terraglio 269 - 31022 PREGANZIOL (TV) - Tel. (0422) 381189 • **VERY IMPORTANT CLIENTS** Via Villa Giusti 58 - 10142 TORINO - Tel. (011) 7070208 • **MEDIA & COMPUTERS** Via Chiusi 76 - 00139 ROMA - Tel. (06) 8110757 • **CM INFORMATICA** Via Canale, 96 - 42013 VILLALUNGA (RE) - Tel. (0522) 841015