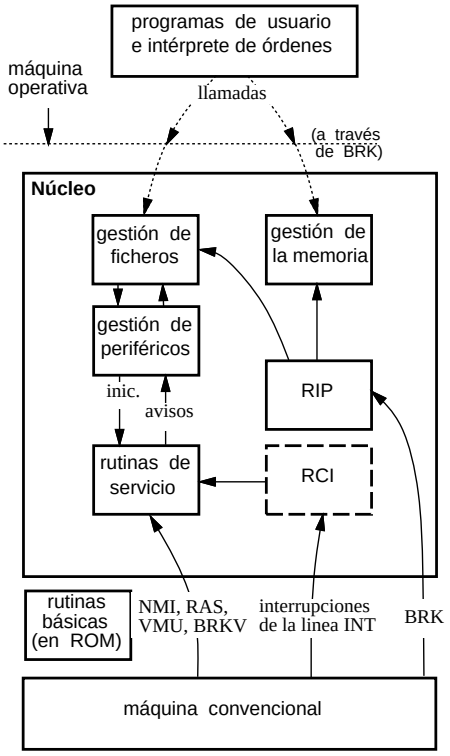
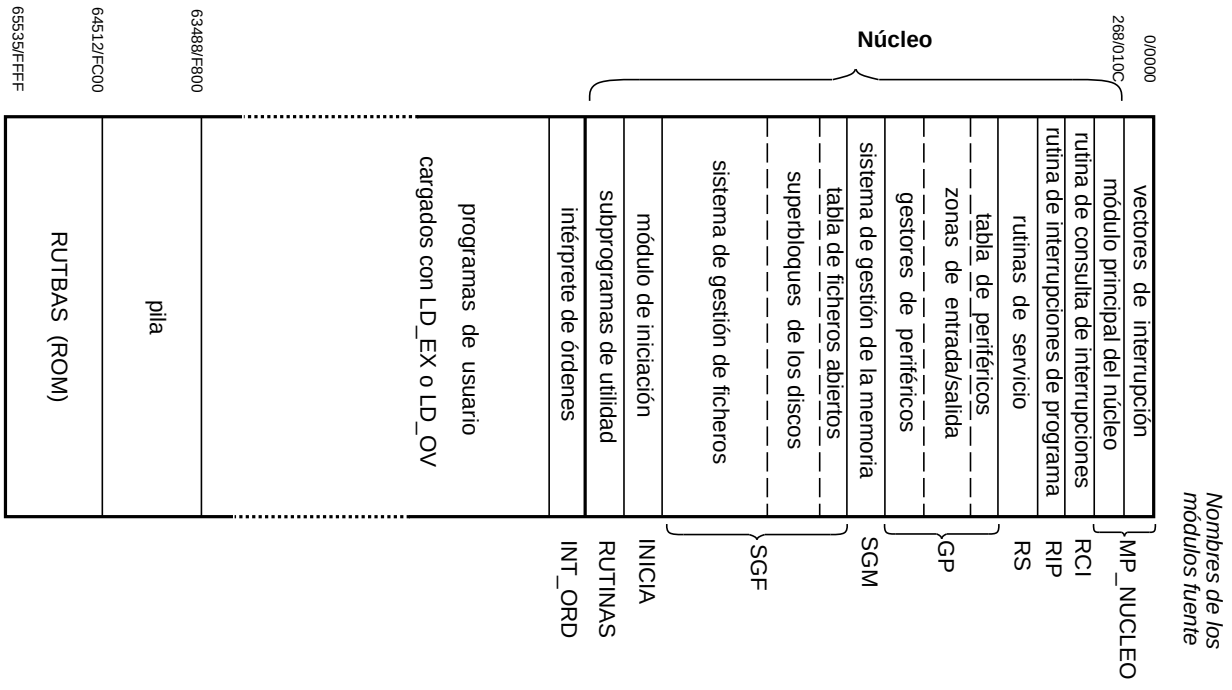
Componentes del núcleo de Monoalgorítmez



Estudiaremos:

1. **Estructuras de datos en la MP** (en forma de **tablas**):
   - punteros y prefijos
   - tabla de periféricos
   - tabla de ficheros abiertos
2. **Algoritmos:**
   - **RIP**
   - **SGM**
   - **SGF**

Mapa de la MP de Monoalgorítmez



Punteros y prefijos

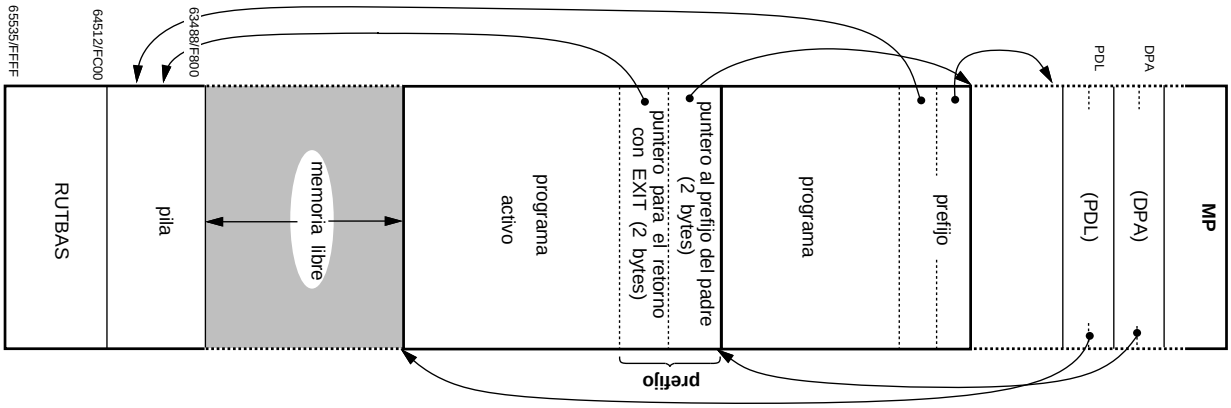


Tabla de periféricos

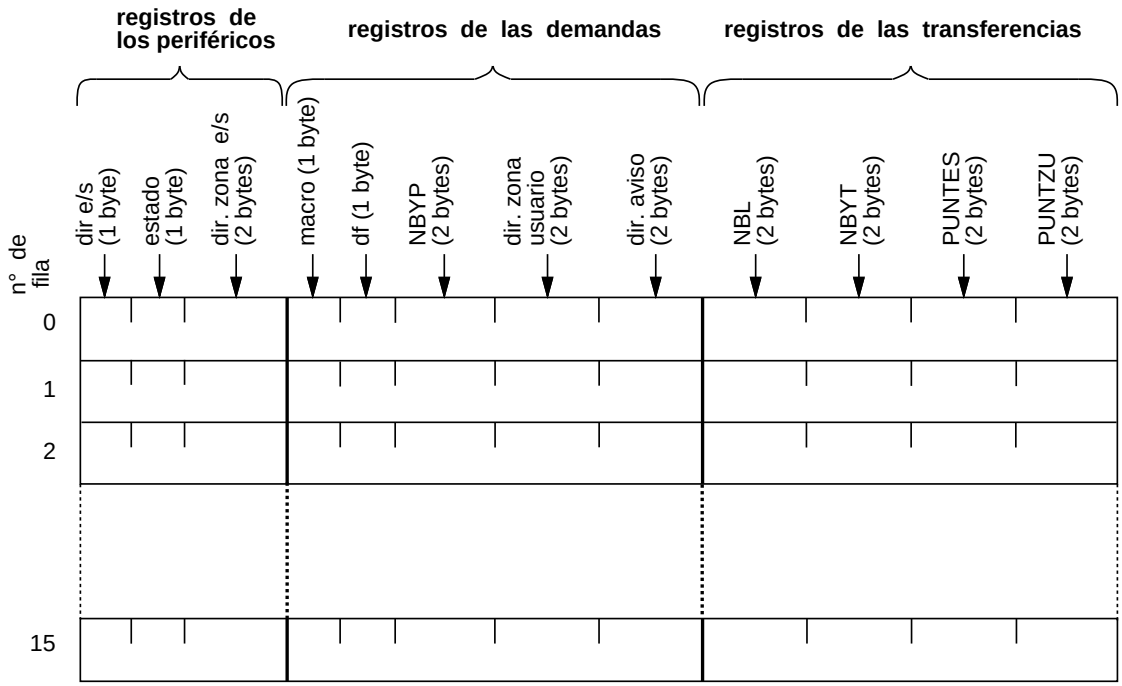
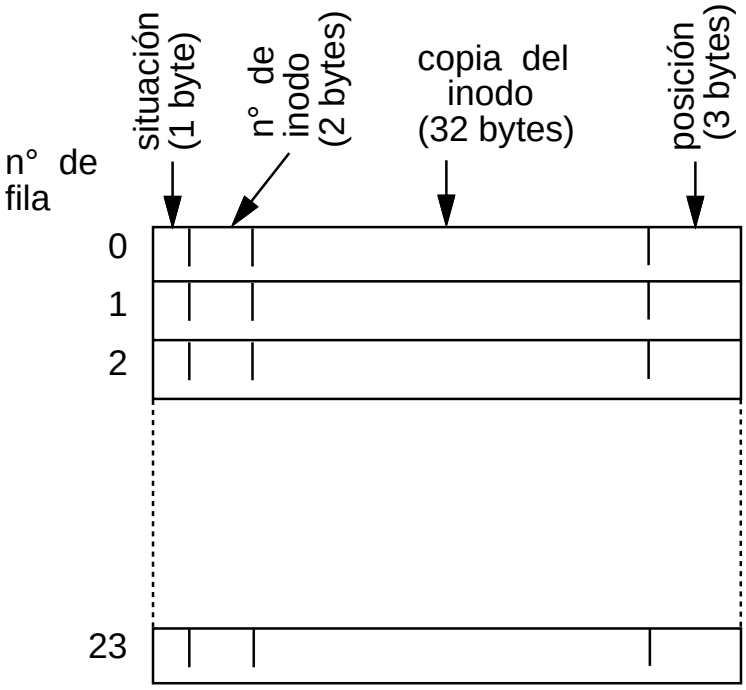


Tabla de ficheros abiertos



La RIP (1)

```
MODULE RIP
  FROM SGM IMPORT SGM_LD_EX, SGM_LD_OV, SGM_EXIT
  FROM SGF IMPORT SGF_CREATE,SGF_DELETE,SGF_OPEN,SGF_CLOSE
  FROM SGF IMPORT SGF_WRITE,SGF_READ,SGF_WRITE_A, SGF_READ_A
  FROM SGF IMPORT SGF_POS, SGF_STAT, SGF_CHMOD
           EXPORT RIP_ENT
TAB_DIR     DATA   SGF_CREATE     ; tabla de direcciones de
            DATA   SGF_DELETE     ; tratamiento de llamadas
            DATA   SGF_OPEN
            DATA   SGF_CLOSE      ; importadas del sistema
            DATA   SGF_WRITE_A    ; de gestión de ficheros
            DATA   SGF_READ_A
            DATA   TR_WAIT_A      ; definida en este módulo
            DATA   SGF_WRITE
            DATA   SGF_READ
            DATA   SGF_POS
            DATA   SGF_CHMOD
            DATA   SGF_STAT
            DATA   SGM_LD_EX      ; importadas del sistema
            DATA   SGM_LD_OV      ; de gestión de la memoria
            DATA   SGM_EXIT
```

La RIP (2)

```
RIP_ENT    CMP.B   .0,#15
           BNN     NODEF ; mayor que 14
           CMP.B   .0,#0
           BN      NODEF ; menor que 0
           CLR     .13   ; no error, de momento
           SHL     .0    ; 2*(R0)=dirección relativa en TAB_DIR
           ADD     .0,#TAB_DIR
           BR      [[.0++]]  ; al tratamiento de la llamada
TR_WAIT_A  LD.B    .0,/0[.5]; (R5) = dirección de "aviso"
           CMP.B   .0,#0
           BNZ     RIP_SAL  ; "aviso" es 1 o -1: retornar
           EI
           WAIT             ; espera interrupción
           DI
           BR      TR_WAIT_A
NODEF      LD.B    .13,#-1  ; error 1: llamada no definida
RIP_SAL    RETI
         END
```


El SGM: tratamiento de LD_EX

1. Busca el nombre (apuntado por R2) en el directorio
2. Comprueba que cabe en memoria:
   (PDL) + longitud + 4 <H'F800
3. Pone puntero a prefijo anterior: (DPA) $\longrightarrow$ (PDL)
4. Pone (PP) en el prefijo: (PP) $\longrightarrow$ (PDL)+2
5. Actualiza DPA y PDL:
   (PDL) $\longrightarrow$ DPA; (PDL) + longitud + 4 $\longrightarrow$ PDL
6. Llama a un cargador reubicador
7. Apila la dirección de ejecución (devuelta por el cargador) y un valor inicial para el registro de estado: H'0100
8. RETI

El SGM: tratamiento de LD_OV y EXIT

- LD_OV es similar, pero
  - no modifica PDL ni DPA ni pone prefijo
  - no apila dirección de ejecución ni RE
  - salva y restaura los registros que utiliza
- EXIT:

```
SGM_EXIT LD   .8, DPA     ; (DPA) → PDL:
         ST   .8, PDL     ;   la nueva PDL es la antigua DPA
         LD   .14,/2[.8]  ; ((DPA)+2) → PP:
                          ;   recupera el PP que dejó LD_EX
         LD   .8, /0[.8]  ; ((DPA)) → DPA:
         ST   .8, DPA     ;   el nuevo programa activo es
                          ;   el padre
         RETI
```


El SGF: creación y borrado

♣ CREATE(#NOMBRE, PERMISOS):

- Si nombre ya existe pone longitud = 0 en el inodo y libera bloques (en mapa de bits)
- Si nombre nuevo, busca un inodo libre, lo marca ocupado, escribe en él los permisos, longitud = 0 y hora, y rellena fila del directorio: número del inodo, nombre del fichero
- Sigue como OPEN en modo escritura (esencialmente, rellena una fila de la tabla de ficheros abiertos)

♣ DELETE(#NOMBRE):

- Libera bloques ocupados («0» en los bits del mapa de bloques)
- Libera el inodo («0» en el bit del mapa de inodos)
- Copia el superbloque ya actualizado en el disco
- Libera la fila del directorio («H'FFFF» en los dos primeros bytes)

El SGF: apertura y cierre

♣ `OPEN(#NOMBRE, MODO)`:

- Comprobaciones previas: nombre existente en el directorio, modo coherente con los permisos del inodo, espacio en la tabla de ficheros abiertos
- Construye la fila de la tabla de ficheros abiertos
- Número de fila de la tabla $\rightarrow$ R13 (descriptor)
- Si es un fichero ordinario y modo 0, 1 o 2, y long$>$0, inicia la lectura de su primer bloque (del disco a la zona de e/s)

♣ `CLOSE(DF)`:

- Comprobación previa: el número pasado por R3 (DF) es el de una fila ocupada de la tabla de ficheros abiertos
- Pone H'FF en ese byte
- Si el fichero se ha modificado, actualiza su inodo en el disco

El SGF: otras llamadas

♣ `POS`, `CHMOD`, `STAT`

Comprobaciones y procesos se deducen fácilmente de las descripciones funcionales

♣ `WRITE_A`, `READ_A`, `WRITE`, `READ`

Procesos más complicados, especialmente para ficheros ordinarios (*abstracción* del medio):

- El SGF (y los gestores) se ocupan de la correspondencia entre el fichero
  y la ubicación física en el disco
- Con `READ` y `WRITE` otra abstracción:
  acceso a un número arbitrario de bytes

Veamos algunos detalles (para ficheros ordinarios)

SGF: preparación de las transferencias

♣ Comprobaciones previas:

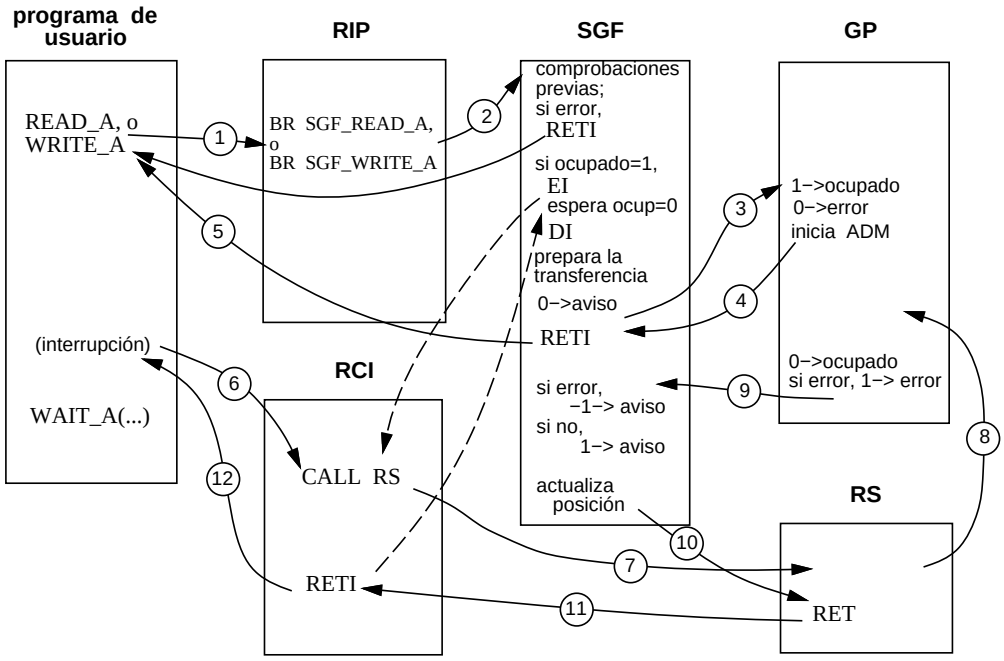
- descriptor de fichero en uso
- operación compatible con modo de apertura
- periférico operativo
- si es `WRITE_A` o `WRITE`: $<$ 133 KB y bloques libres en disco
- si es `READ_A` o `READ`: posición $<$ longitud

♣ Preparación de la transferencia:

parámetros de la macro $\longrightarrow$ registro de la demanda:

- número de la macro, descriptor del fichero
- número del bloque (se obtiene del inodo, y éste, del descriptor)
- si es `READ_A` o `WRITE_A`, direcciones de «aviso» y de zona usuario; 512 $\longrightarrow$ NBYP
- si es `READ` o `WRITE`, dirección de zona usuario; número de bytes a NBYP

Procesos para `READ_A` y `WRITE_A`



- SGF: fichero y posición $\Rightarrow$ número de bloque
- gestor: número de bloque $\Rightarrow$ superficie, pista y sector

READ y WRITE: doble «buffering»

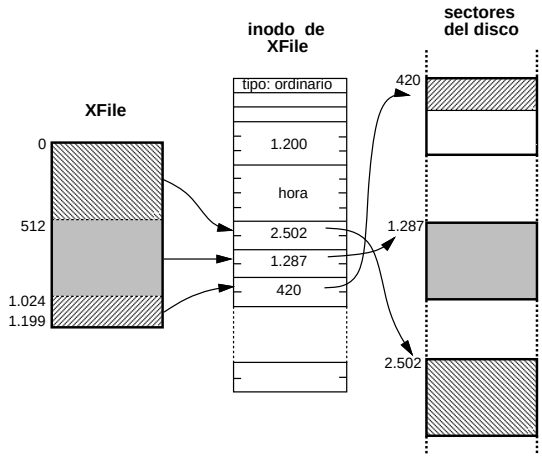
- transferencia de un número arbitrario de bytes ($<$ 32 KB)
- «vigilancia» del fin de la transferencia transparente al programador
- zona de usuario y zona de e/s (512 bytes para ficheros ordinarios)
- WRITE:
  - NBYP = número total de bytes
  - Mientras NBYP $>$ 0:
    - calcular NBYT:
      - si *pos* mód 512 = 0 y NBYP $<$ 512, NBYT = NBYP;
        si *pos* mód 512 = 0 y NBYP $\geq$ 512, NBYT = 512
      - si *pos* no está al comienzo de un bloque es algo más complicado
    - NBYP = NBYP - NBYT
    - si periférico ocupado, espera
    - si NBYT $<$ 512, lee bloque del disco a la zona de e/s
    - copia NBYT bytes de la zona de usuario a la de e/s
    - el gestor inicia la escritura de la zona de e/s en el bloque
    - actualiza posición
    - si NBYP = 0 se devuelve control al programa de usuario (RETI)

READ: anticipación

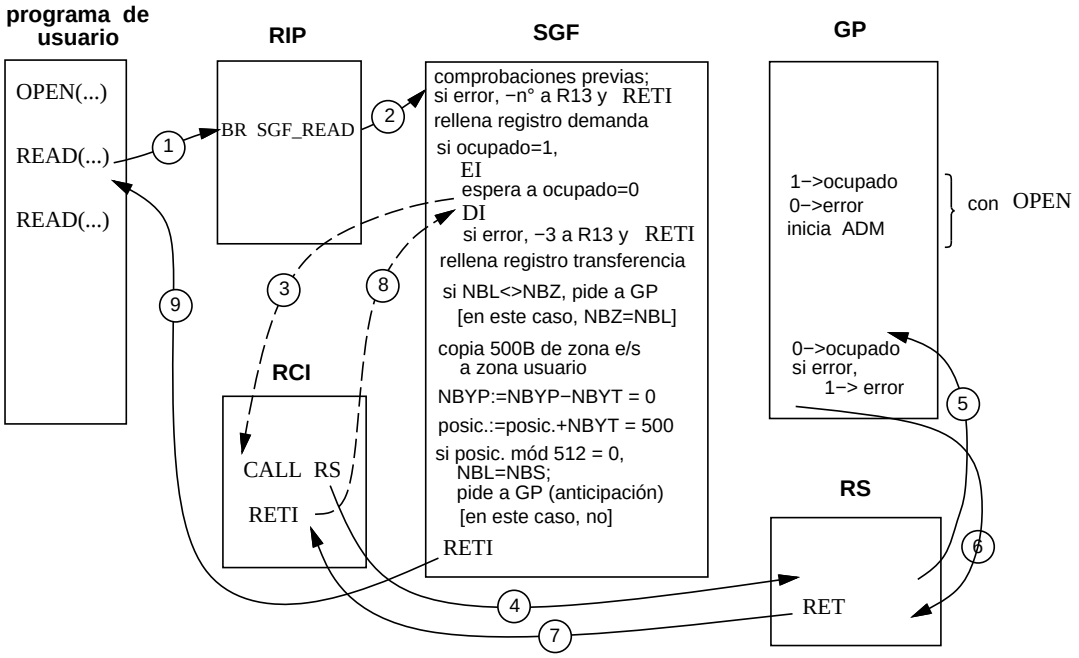
- OPEN inicia ya la transferencia del primer bloque a la zona de e/s (si longitud $>$ 0 y modo 0, 1 o 2)
- En el gestor, dos variables: NBZ = número del bloque en la zona de e/s y NBS = número del siguiente (en el inodo)
- Al terminar una transferencia con READ, si posición es múltiplo de 512 se inicia la lectura del bloque NBS
- Al empezar una transferencia, si NBL (número del bloque a leer) = NBZ no hace falta leer
- *Observación*: anticipación muy limitada; sería necesaria, al menos, una segunda zona de e/s que contuviese el bloque NBS

Ejemplo con READ: planteamiento

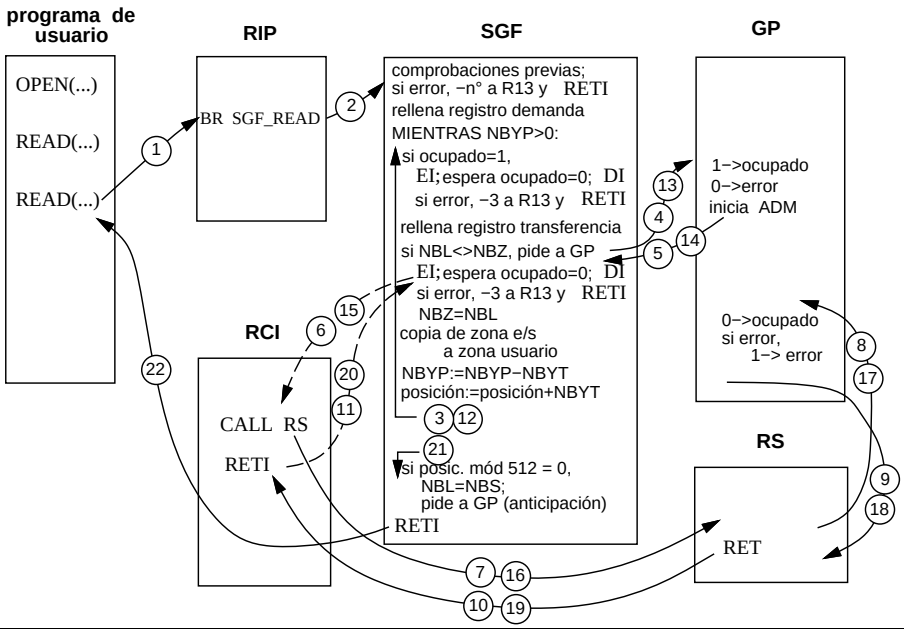
```
        ---
NOMFICH DATA.B  "A:XFile",0
ZU1     RES.B   500
ZU2     RES.B   600
DFICH   RES.B   1
        ---
        OPEN(#NOMFICH,#2)
        ST.B   .13, DFICH
        ---
        READ(DFICH,#ZU1,#500)
        ---
        READ(DFICH,#ZU2,#600)
        ---
```



Ejemplo con READ: la primera llamada



Ejemplo con READ: la segunda llamada



	NBYP	NBL	NBYT	PUNTES	PUNTZU
2–3	600→588	2.502	12	dir._zona e/s+500	dir._zona_usuario
4–12	588→76	1.287	512	dir._zona e/s	dir._zona_usuario+12
13–21	76→0	420	76	dir._zona_e/s	dir._zona_usuario+524

Monoalgorítmez, MS–DOS y UNIX (1)

- De MS–DOS:
  - monoprogramación
  - `A:`, `B:`, ...
  - «prefijos» (en MS–DOS, PSPs: 256 bytes)
  - mapa de la MP, con vectores en direcciones bajas y ROM en altas
- De UNIX:
  - órdenes del intérprete
  - inodos, superbloques y mapas de bits
  - núcleo monolítico incluyendo gestores

Monoalgorítmez, MS–DOS y UNIX (2)

- Llamadas al sistema:

Monoalgorítmez	UNIX	MS–DOS
CREATE	creat	create
DELETE	unlink	delete
OPEN, CLOSE	open, close	open, close
WRITE_A, READ_A		
WAIT_A		
WRITE, READ	write, read	write, read
LD-EX	fork/exec	load-and-exec
LD-OV		load-and-overl
EXIT	exit	end-prog
POS	lseek	pos
CHMOD	chmod	chmod
STAT	stat	get-file