Multialgorítmez

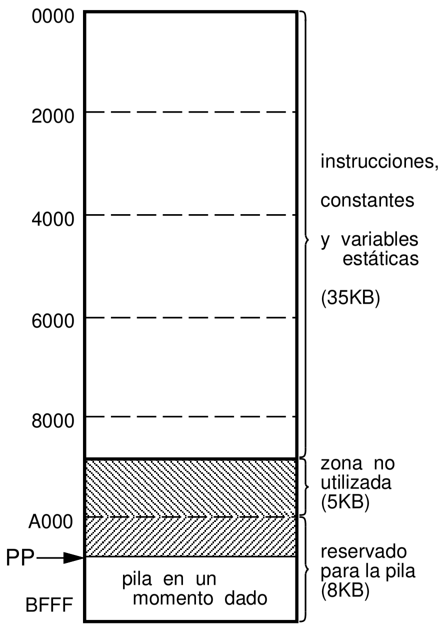
1. Características:
   - Características funcionales
   - Procesos y gestión de la memoria
2. Modelo funcional:
   - Acceso e interfaz de usuario
   - Llamadas al sistema
3. Modelo estructural:
   - Estructuras de datos del núcleo
4. Modelo procesal:
   - Inicio
   - Extensión de algoritmos de RIP, SGM y SGF
   - Distribuidor y planificador

Características de Multialgorítmez

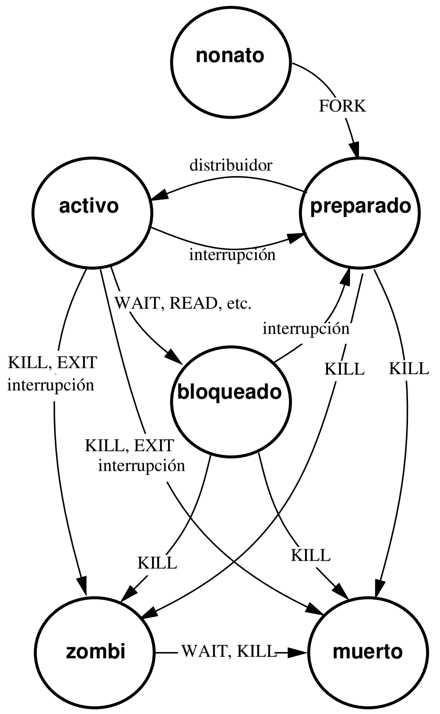
Sistema multiusuario y multitarea

⇒ concepto de **proceso** fundamental

- $1 \leq$ pid ≤ 64; proceso nulo: pid = 0
- **No memoria virtual**
  (espacio direcc.: 64 KB...)
- **Memoria física de 1 MB**
  gracias a la UGM
- **Pila** del núcleo
  y una pila por proceso
- **Imagen en memoria** de un proceso:
  (2–8 páginas de 8 KB)



Estados y transiciones de un proceso



Cambios de contexto (1)

(1) De un proceso de usuario al SO:

- Causado por una interrupción
- Actividades:

1. (PP)–2→PP; CP→(PP); (PP)–2→PP; RE→(PP)
2. 0 → PIN, 1 → SUP. Esto hace modo = 1, lo que provoca que:
3. La UGM pone los valores del núcleo en los registros de la tabla de páginas
4. La UCP lee vector de interrupción y lo pone en CP
5. Se ejecuta la rutina, que empieza salvando los registros R0 – R14 (en la «tabla de procesos») y poniendo en R14 un valor para la pila del núcleo

Cambios de contexto (2)

(2) Del SO a un proceso de usuario:

- Se realiza en el distribuidor
- Actividades

1. Se cargan direcciones de página para ese proceso en la UGM
2. Se restauran los registros R0 – R14
3. Se ejecuta `RETI`:
   - $0 \longrightarrow$ SUP; UGM carga nuevos valores en los registros de la tabla de páginas
   - $((PP)) \longrightarrow RE; (PP) + 2 \longrightarrow PP;$
   - $((PP)) \longrightarrow CP; (PP) + 2 \longrightarrow PP;$

Modelos a estudiar:

♣ Modelo funcional

- acceso al sistema (autenticación)
- intérprete de órdenes (interfaz del usuario)
- llamadas al sistema (interfaz del programador)

♣ Modelo estructural

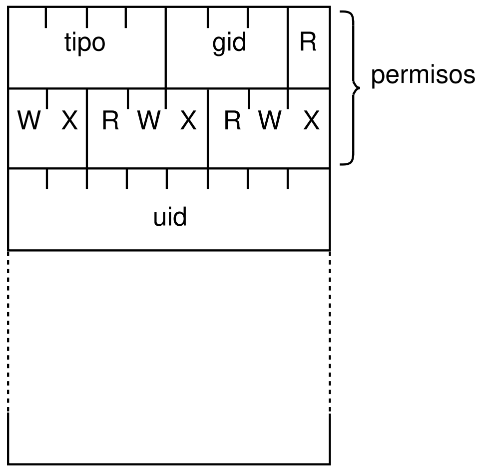
- tabla de procesos y cola del procesador
- tabla de periféricos y colas de demandas
- tabla de posiciones de ficheros
- mapas de bits y fichero de contraseñas

♣ Modelo procesal

- inicialización (creación de cuatro máquinas virtuales)
- RIP, SGM y SGF
- distribuidor y planificador

Acceso al sistema

- identificadores de usuario («**uid**») (0–255)
- identificadores de grupo («**gid**») (0–7)
- administrador, o **superusuario** (uid=0)
- autenticación:
  - contraseñas cifradas
  - mecanismo `login/password`
  - fichero `psswd`
- bits de permiso en los inodos:



Intérprete de órdenes

♣ Además de las órdenes de Monoalgorítmez, `logout`, `passwd`, `ps`, `kill <pid>`, `nice <n> <orden>`, `fg`, `bg`

♣ Ejecución
- **en escena** *(foreground)*
- **entre bastidores** *(background)*

- `progr &`: procesa `progr` entre bastidores
- `Ctrl-z`: bloquea proceso en escena
- `Ctrl-c`: mata proceso en escena
- `fg`: activa (en escena) proceso bloqueado con `Ctrl-z`
- `bg`: activa (entre bastidores) proceso bloqueado con `Ctrl-z`

Llamadas al sistema

Las de Monoalgorítmez (salvo `LD_EX` y `LD_OV`), y:

- `FORK`: crea un proceso hijo (imagen idéntica a la del padre; el padre recibe el pid del hijo en R13, y el hijo recibe «0» en R13)
- `EXEC(#NOM,#ARG,LONG)`: sustituye la imagen en memoria por la de `NOM` (el proceso sigue siendo el mismo)
- `WAIT(PID)`: bloquea al proceso hasta que termina el hijo que tiene pid=(`PID`); si `WAIT(#0)`, el primer hijo que termine lo desbloquea y comunica su pid por R13
- `KILL(PID)`
- `SETPRIORITY(PID, BASE)` (0≤`BASE`≤10)

Ejemplo de `FORK/EXEC` (1)

```
      FORK   ; tras su ejecucion, DOS COPIAS IDENTICAS EN LA MP
      CMP.B  .13, #0
      BZ     HIJO
      ---    ; instrucciones para el proceso padre
      EXIT
HIJO  ---    ; instrucciones para el proceso hijo
```

Generalmente, el hijo empieza con `EXEC`:

```
PID   RES.B    1
PROG  DATA     "A:un_programa",0
ARG   DATA     100; o los argumentos que sean
      ---
      FORK
      CMP.B    .13, #0
      BZ       HIJO
      ST.B     .13,PID
      ---       ; puede seguir haciendo otras cosas
      WAIT(PID); necesita que haya terminado el hijo
      ---
HIJO  EXEC(#PROG,#ARG,#2)
```

Ejemplo de FORK/EXEC (2)

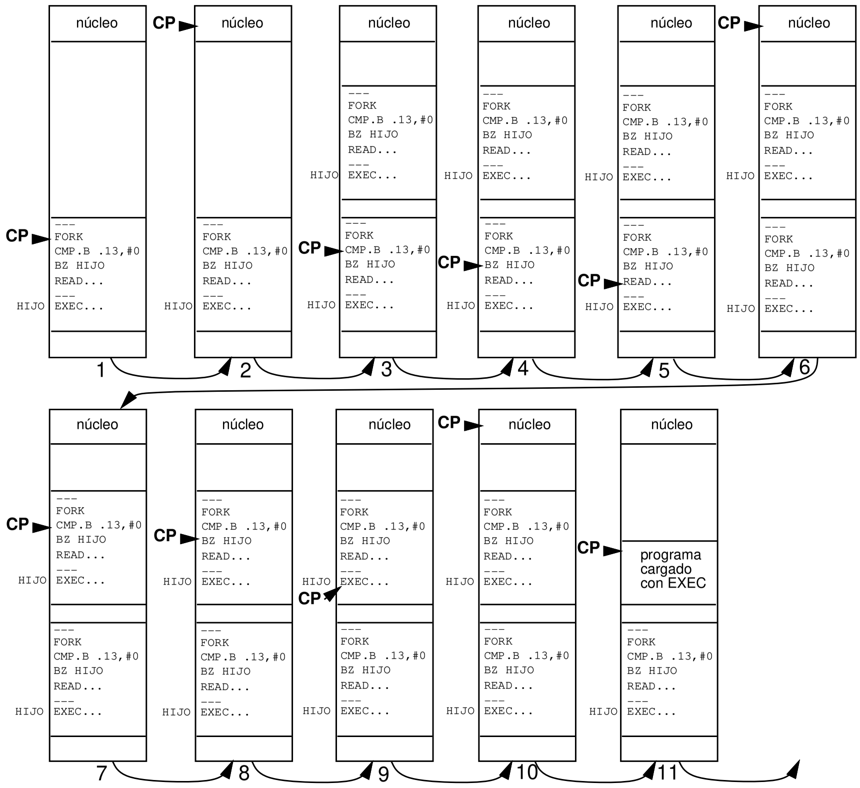
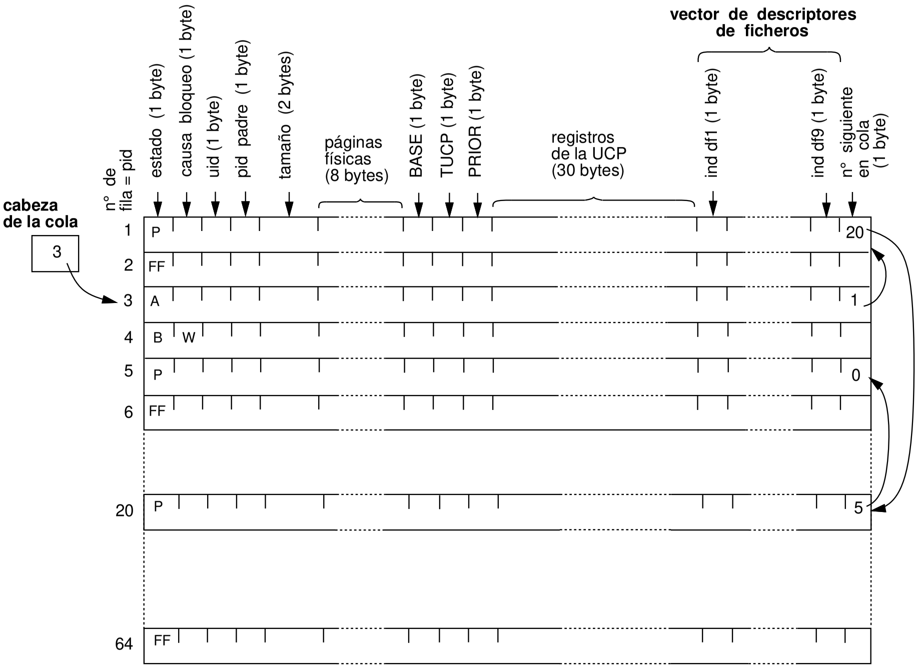
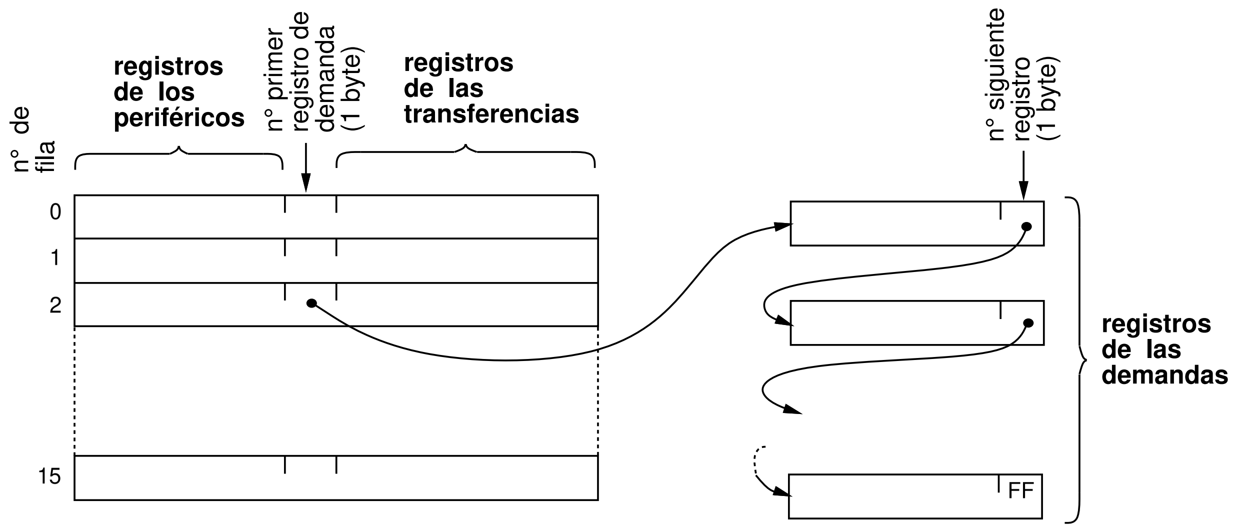


Tabla de procesos y cola del procesador



Colas de demandas de periféricos



Tablas de periféricos y de demandas

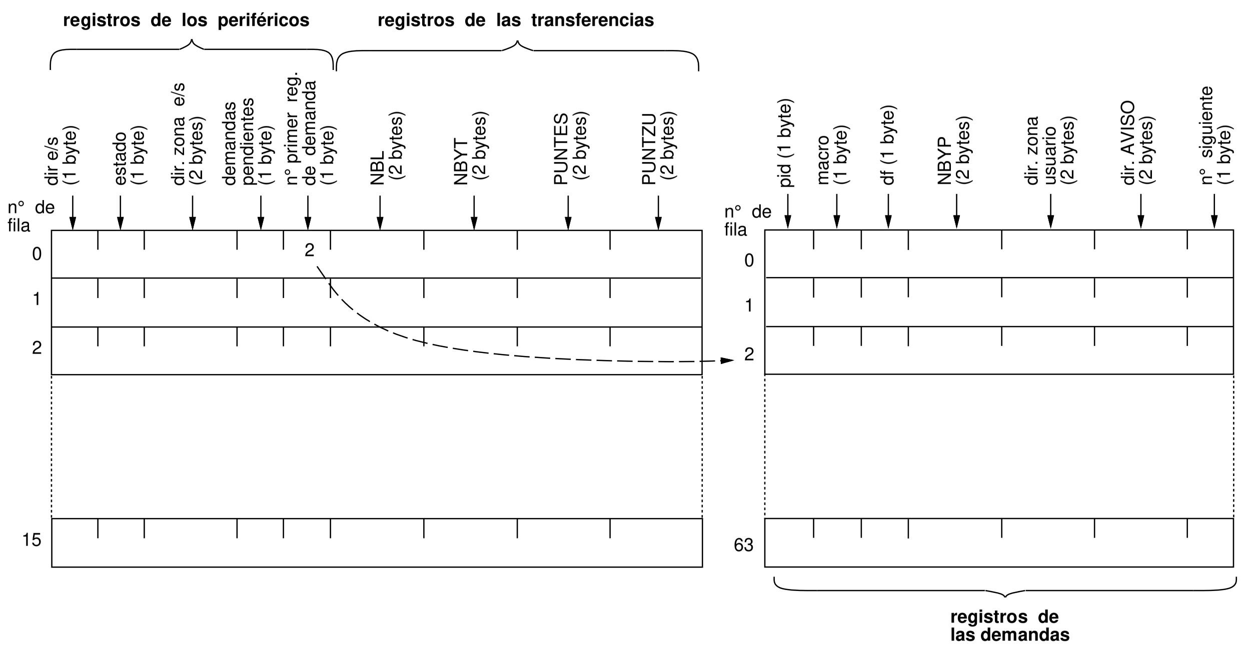
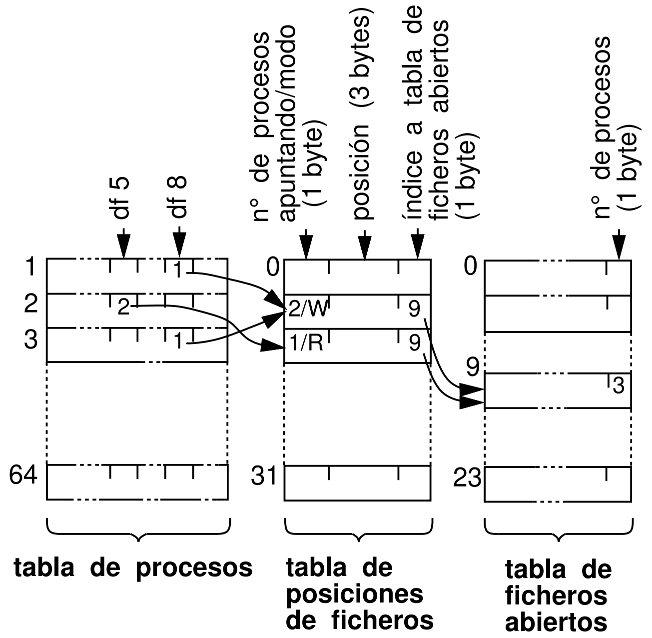


Tabla de posiciones de ficheros



Mapas de bits y fichero de contraseñas

♣ Mapas de bits

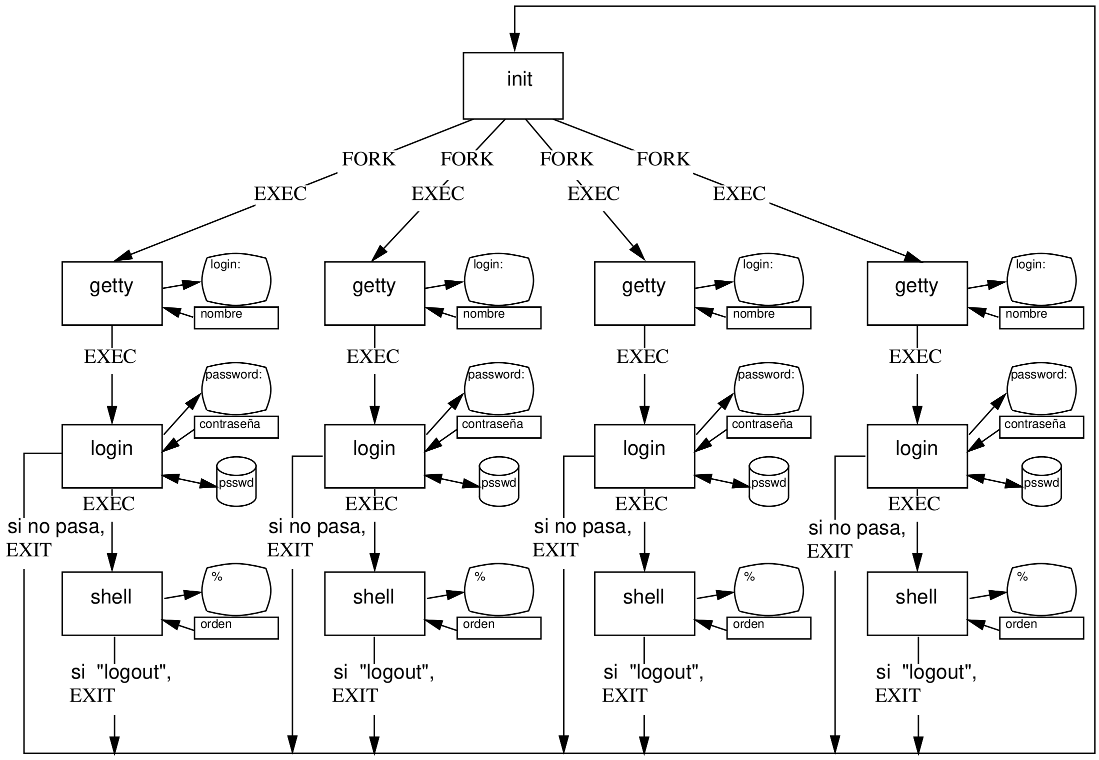
- de inodos y de bloques:
  como en Monoalgorítmez
- de páginas físicas libres (sólo en la MP)

♣ Fichero de contraseñas

256 registros con estos campos:

- uid y gid
- login
- contraseña (cifrada) (se puede cambiar con `passwd`)
- datos del usuario (se puede cambiar con `chfn`)
- intérprete de órdenes (se puede cambiar con `chsh`)

Modelo procesal general



El programa `init` (1)

```
MODULE INIT
EXPORT INIT_ENT
  PID0     RES.B    1
  PID1     RES.B    1
  PID2     RES.B    1
  PID3     RES.B    1
  NGETTY   DATA.B   "A:getty",0
  NTTY0    DATA.B   0
  NTTY1    DATA.B   1
  NTTY2    DATA.B   2
  NTTY3    DATA.B   3
INIT_ENT FORK
         CMP.B  .13,#0
         BZ     TTY0
         ST.B   .13,PID0
         FORK
         CMP.B  .13,#0
         BZ     TTY1
         ST.B   .13,PID1
         FORK
         CMP.B  .13,#0
         BZ     TTY2
         ST.B   .13,PID2
         FORK
         CMP.B  .13,#0
         BZ     TTY3
         ST.B   .13,PID3
SIGUE    WAIT(#0)
         --- ;  (2)
TTY0     EXEC(#NGETTY,#NTTY0,#1)
TTY1     EXEC(#NGETTY,#NTTY1,#1)
TTY2     EXEC(#NGETTY,#NTTY2,#1)
TTY3     EXEC(#NGETTY,#NTTY3,#1)
       END
```

El programa `init` (2)

```
SIGUE   WAIT(#0)                F0    FORK
        CMP.B   .13,PID0              CMP.B   .13,#0
        BZ      F0                    BZ      TTY0
        CMP.B   .13,PID1              ST.B    .13,PID0
        BZ      F1                    BR      SIGUE
        CMP.B   .13,PID2        F1    FORK
        BZ      F2                    CMP.B   .13,#0
        CMP.B   .13,PID3              BZ      TTY1
        BZ      F3                    ST.B    .13,PID1
        BR     SIGUE                  BR      SIGUE
                                F2    FORK
                                      CMP.B   .13,#0
                                      BZ      TTY2
                                      ST.B    .13,PID2
                                      BR      SIGUE
                                F3    FORK
                                      CMP.B   .13,#0
                                      BZ      TTY3
                                      ST.B    .13,PID3
                                      BR      SIGUE
```


La RIP: diferencias con Monoalgorítmez

- Al empezar, salva el *estatus* del proceso activo (R0–R14 al registro de la tabla de procesos)
- (`PPNUCLEO`) $\longrightarrow$ R14 (normalmente, H'FC00)
- Al terminar, (R14) $\longrightarrow$ `PPNUCLEO` y en lugar de `RETI`, `BR DISTRIB`
- Tratamiento de `WAIT_A(#AV)`: si `AV`=0 bloquea al proceso (no `WAIT`)
- Tratamiento de `WAIT(PID)`: considera estado del proceso `PID`:
  - si el proceso `PID` no está zombi, bloquea al proceso (padre)
  - si está zombi, lo mata, modo fin $\longrightarrow$ R12 y deja preparado al padre
- Tratamiento de `SETPRIORITY(PID,BASE)`: comprueba derechos y modifica `BASE` en el registro de `PID` de la tabla de procesos
- Para `FORK`, `EXEC`, `EXIT` y `KILL` llama al SGM
- Para `CHMOD`, `READ`, etc., llama al SGF

El SGM: Tratamiento de `FORK` y `EXEC`

♣ `FORK`: Si hay sitio en la tabla de procesos y en la MP,

1. Asigna páginas y copia la imagen del proceso activo
2. Actualiza tabla (y cola) de procesos y mapa de bits de páginas
3. Rellena registro del nuevo proceso
4. En el lugar para R13 del nuevo registro pone «0», y en el del padre pone el número del nuevo

♣ `EXEC`: Si hay sitio en la MP y si hay permiso (X = 1),

1. Libera páginas o asigna nuevas
2. Coge los argumentos de `EXEC`
3. Copia el fichero en las páginas asignadas
4. Limpia la pila del proceso y pone en ella los argumentos y valores iniciales de CP y RE
5. Actualiza registro del proceso y mapa de bits de páginas

El SGM: Tratamiento de `EXIT` y `KILL(PID)`

1. Si `KILL`, comprueba previamente el *uid*
2. Pone codificación sobre la causa de terminación (`EXIT` o `KILL`) en el lugar reservado para R0 (`WAIT` del padre: se copia en R12)
3. Si padre esperando lo desbloquea (preparado), pone PID en R13 del padre, libera el registro, lo saca de la cola y actualiza ésta
4. Si no, se hace zombi y lo saca de la cola
5. Libera las páginas ocupadas
6. Si tiene hijos zombis, los mata («libre» en «estado»)
7. Si quedan huérfanos (hijos que no han terminado su tarea) los deja, pero los marca como hijos de `init`

El SGF

Se ocupa de las llamadas para creación, borrado, apertura, cierre, lectura, escritura, `POS`, `STAT` y `CHMOD`

Diferencias más importantes con Monoalgorítmez:

- Comprobaciones previas sobre permisos de acceso
- `READ`, `WRITE`, etc.: si periférico ocupado, bloquea al proceso y pone la demanda en cola
- `OPEN`: si fichero ya abierto y no compartible, bloquea al proceso
- `CLOSE`:
  - decrementa «número de procesos» en la tabla de ficheros abiertos
  - si resulta «0» y hay procesos bloqueados por `OPEN` pone «1» y desbloquea al más prioritario
  - si resulta «0», no hay proceso esperando, el fichero era ordinario y el inodo se ha modificado actualiza en disco la copia del inodo

El distribuidor

Se bifurca a él desde la RIP o desde la RCI o desde una rutina de servicio, que han actualizado la cola del procesador

1. Marca «activo» al primer proceso de la cola
2. Prepara en la UGM la función de correspondencia para este proceso que se va a activar
3. Restaura en los registros R0–R14 de la UCP los valores que encuentra en el registro de este proceso
4. `RETI` (el hardware cambia automáticamente la función de correspondencia, y los valores que se recuperan de CP y RE son los de la pila del proceso)

El planificador (1)

♣ **Prioridad** (campo `PRIOR` de la tabla de procesos):

- entre 0 *(máxima)* y 19 *(mínima)*
- suma de un valor constante («base») y otro dinámico («TUCP»)

♣ **Prioridad de base y tiempo «reciente» de UCP** (campos `BASE` y `TUCP` de la tabla de procesos):

- `BASE`: entre 0 *(alta)* y 10 *(baja)* (5: media)
- Cada 100ms (interrupción de reloj) se incrementa el TUCP *del proceso activo*
- Cada segundo, *todos* los TUCPs se dividen por dos, se recalcula `PRIOR=BASE+TUCP` y se actualiza la cola del procesador

El planificador (2)

Parte de la rutina de servicio del reloj:

```
para el proceso activo: TUCP:=TUCP+1;
K:=K+1;
si K=10 entonces
     K:=0;
     para todos los procesos:
        TUCP:=shr(TUCP);
        PRIOR:=BASE+TUCP;
actualizar la cola del procesador;
BR DISTRIB
```