Está Coverflow - Fé de erratas

- En el archivo de configuración de la UMV falta el tamaño del bloque de memoria que funcionará como memoria principal.
- En el archivo de configuración del Kernel falta la IP/Puerto de la UMV.
- El enunciado del TP dice que el Stack tiene un tamaño fijo de 100 bytes. Lo correcto es que el tamaño del Stack sea configurable en el archivo de configuración del Kernel.
- En el archivo de configuración del Kernel faltan las variables globales.
- El “Diagrama de Estados de un proceso en el sistema” tiene una transición de más desde Ready hacia el PLP. El Diagrama corregido es el siguiente:

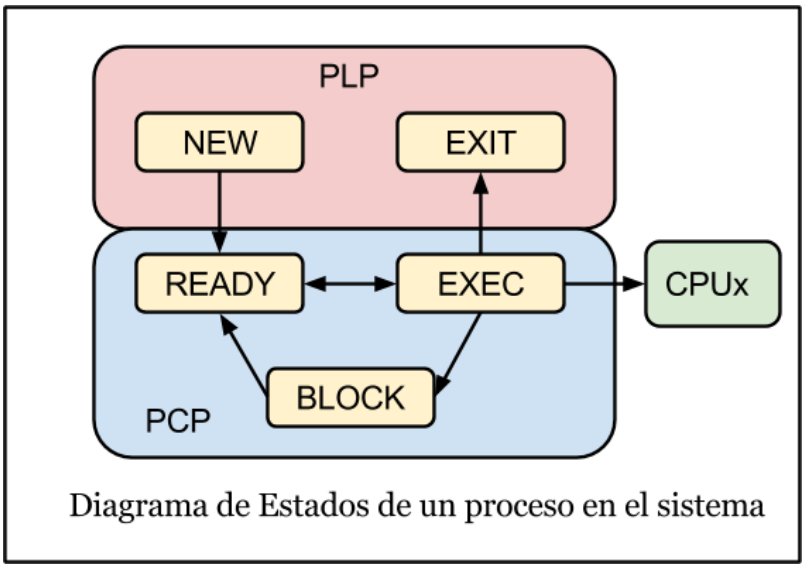

Diagrama de Estados de un proceso en el sistema

- El PLP pondrá a todos los Programas recién creados en la cola de New y, en la medida en que el Grado de Multiprogramación lo permita, moverá los Programas de New a Ready, seleccionando según el algoritmo SJN.
- En el enunciado lo nombra poco, así que lo reafirmamos: efectivamente, el PCP planifica según el algoritmo Round Robin.
- En la lista de tareas a realizar del PCP dice:

  > Recibirá los PCB del PLP y los encolará en la cola de READY, según el algoritmo de planificación de corto plazo Round Robin.

  Dado que es el PLP el encargado de transicionar Programas de New a Ready, esta tarea debería decir:

  > Planificará según el algoritmo Round Robin a los procesos encolados en la cola de READY.

- Página 7, `System Calls`:

  > En el código Ansisop el identificador de la variable compartida, para diferenciarlo de las variables normales, comenzará con el caracter signo de admiración (!), seguido del identificador de un caracter alfabético, por ejemplo: !a, !g, !q

  La restricción de un único caracter alfabético es incorrecta. Las variables globales se identifican con un `!` y luego una cadena alfanumérica, como correctamente dice la página 18. Por ejemplo, `!a`, `!A`, `!compartida`, `!ParaTodos`, `!Compartidas10`.

- El PCB debe incluir un campo numérico con el tamaño del segmento Índice de Etiquetas, a fin de que la CPU pueda obtenerlo de la UMV para la búsqueda de etiquetas.
- Estructura del PCB: aclaramos el tipo de cada dato.

Estructura	Tipo	Descripción

Identificador único	Numérico	Identificador único del Programa en el sistema
Segmento de código	Dirección	Dirección del primer byte en la UMV del segmento de código
Segmento de stack	Dirección	Dirección del primer byte en la UMV del segmento de stack
Cursor del stack	Dirección	Dirección del primer byte en la UMV del Contexto de Ejecución Actual
Índice de código	Dirección	Dirección del primer byte en la UMV del Índice de Código
Índice de etiquetas	Dirección	Dirección del primer byte en la UMV del Índice de Etiquetas
Program counter	Numérico	Número de la próxima instrucción a ejecutar
Tamaño del Contexto Actual	Numérico	Cantidad de variables (locales y parámetros) del Contexto de Ejecución Actual
Tamaño Índice de etiquetas	Numérico	Cantidad de bytes que ocupa el Índice de etiquetas

- Las primitivas `llamarSinRetorno` y `llamarConRetorno` figuran en el enunciado con un parámetro `t_puntero_instruccion linea_en_ejecuccion` que no aparece en la documentación del Parser. Efectivamente ese parámetro se eliminó de la interfaz, y **no deberá usarse**. Como nota general, ante una incongruencia entre la documentación del Parser y la especificación de AnSISOP que figura como Anexo del TP, la documentación del Parser prevalecerá.
- La CPU deberá obtener por archivo de configuración con la IP y Puerto del Kernel y los de la UMV.
- Al pie de la página 13, el enunciado dice :

  > si el Programa B solicita 5 bytes iniciando en la posición 25 (15 bytes de offset del inicio del segmento 2)

  El ejemplo es incorrecto, dado que la UMV especifica que los pedidos deben hacerse especificando base del segmento y offset por separado. En los casos citados, debería hacer pedidos de 5 bytes con offset 15 a partir de la base 10 (`leer(10, 15, 5)`).
- En la UMV, la operación de handshake dice incluir un parámetro Identificador del Programa. Ese dato sobra, y no deberá ser enviado. La operación handshake sólo tiene como parámetro el tipo de proceso: Kernel o CPU.
- En la sección "System Calls", en la página 7, el enunciado dice que sólo existen 5 system calls: `obtener_valor` , `grabar_valor` , `wait` , `signal` y `entrada_salida` . En realidad existen otras dos más, como se especifica en el Anexo V: `imprimir` e `imprimirTexto` .
- En la sección "Fin de la ejecución", en la página 12, el enunciado dice que *"el CPU deberá solicitar la destrucción de todas las estructuras correspondientes [...] y notificarle al PCP que el proceso finalizó"*. En realidad, no es responsabilidad directa del CPU solicitar la destrucción de las estructuras correspondientes: la CPU debe hacer lo necesario para que quien tenga la responsabilidad de destruir las estructuras se entere de que es momento de hacerlo.
- En el enunciado, seccion UMV, consola, donde dice *"dump: Este comando generará un reporte en pantalla y opcionalmente en un archivo en disco."* Deberia decir "dump: Este comando generará un reporte en un archivo en disco, escribiendo el contenido siempre al final del mismo (append)."
- La UMV no logueará por pantalla, escribira unicamente en un archivo de log. Esto es para evitar confusiones al momento de escribir comandos en la consola de la misma. Unicamente se escribirán por pantalla los resultados de dichos comandos (a excepcion del comando "dump", que siempre escribe el reporte en un archivo aparte en disco).
- En el Kernel se debe poder visualizar en todo momento el estado de las distintas colas (nuevos, listos, bloqueados - i/o, semaforos-, ejecutando, finalizados).