Simulación de Eventos Discretos

Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Trabajo práctico Nº 1:

Simulación de un Sistema Operativo y evaluación de algoritmos de planificación.

Mariano Zapatero

LU 547 / 95

Mayo de 2008

Indice

Introducción

Objetivo el trabajo

El objetivo de este trabajo es la simulación del funcionamiento de un sistema operativo y la evaluación del rendimiento de distintos algoritmos de planificación de procesos.

Consideraciones generales

Para este trabajo, se simplificarán tanto la estructura como las acciones que realiza un sistema operativo, focalizándose en los tiempos que un proceso ingresa, permanece y sale de la cola de listos. Por lo tanto no se incluirán todos los componentes de un sistema operativo, ni sus tablas, ni rutinas.
Se hará que la CPU, analizando un proceso, sepa qué debe realizar, aun sin tener instrucciones dentro de los mismos. Para esto se codificará una cantidad de entradas/salidas (indistintas en este trabajo) y una cantidad de tiempo de procesador para cada proceso.
A la vez, se asumirá que a ninguno de los modelos le llegan dos mensajes simultáneos.
Otra consideración importante es que no se tuvieron en cuenta optimizaciones de los algoritmos ni de las estructuras utilizadas.

Herramientas y entorno de desarrollo

Para el desarrollo del trabajo se utilizó el plugin CD++ para Eclipse. Se intentó instalar en distintas versiones del programa bajo la plataforma Linux y no fue posible por diversos motivos (algunos casos todavía no se sabe cuál fue la causa). Por lo tanto, se lo utilizó desde una máquina virtual (qemu/virtualbox) con Windows, instalando la versión completa (que incluye Eclipse 2.1, Cygwin, etc.) bajada desde el sitio provisto por la cátedra. Si bien fue posible desarrollar sobre esta plataforma, el proceso fue mucho más lento e incluyó problemas propios del armado de dicho entorno sumados a la pérdida de rendimiento natural de dichas máquinas virtuales.

Descripción del modelo elegido

El modelo top se compone de tres modelos (S.O., CPU y Reloj) separando en primera instancia los distintos componentes de hardware (CPU y Reloj) del componente software (S.O.).
Recibe una única entrada (la entrada de un trabajo) y genera una única salida (el fin de un trabajo).

Las funcionalidades de la CPU serán acotadas de la siguiente manera:
el modelo ejecutará un algoritmo que indicará qué debe hacer el proceso que "utiliza" la CPU, pudiendo ser "Procesar", "Hacer Entrada/Salida" o " Terminar". La primera opción indica que el proceso sólo requiere la CPU por lo que ejecutará hasta que sea desalojado (si la planificación es con desalojo, como lo son las utilizadas en este trabajo). Las dos últimas generarán una salida del modelo indicando la acción a realizar con dicho proceso.
A la vez, el modelo tendrá además las entradas "ejecutar", que indica la entrada de un proceso a la CPU y "tiempo", que indica que se cumplió el *quantum* asignado al proceso y debe ser desalojado. Esta última entrada, a su vez, genera una nueva salida que indica que se hace efectivo el desalojo del proceso.

El modelo Reloj admite como entradas la cantidad de tiempo que debe controlar y genera una única salida indicando que se cumplió dicho tiempo.

Mientras tanto, el modelo S.O. será explotado en distintos sub-modelos:

- AdministradorTrabajos: es el encargado de recibir los *trabajos* del "exterior" y transformarlos en *procesos*.
- BloqueadoE/S: es donde esperarán los procesos que requieran una Entrada/Salida (simulando el tiempo que demoraría el periférico en hacer la tarea).
- Listos: es la cola de listos, es decir, los procesos que están esperando al procesador para ejecutarse. De ellos se elegirá a uno para que pase a la CPU. La forma de elección será dada por el algoritmo de planificación.
  Este es el punto central de este trabajo, ya que lo que se intenta es evaluar el rendimiento de dichos algoritmos.
- Coordinador: este modelo no se corresponde a ninguno de un sistema operativo real, si no que engloba a varias rutinas y módulos del mismo. Para no complejizar en demasía el modelo y, ya que el objetivo del trabajo está fuera de estos y su actividad se reduce prácticamente a pasar mensajes, fueron unidos en único módulo. Aquí se representan, por ejemplo, el controlador de tráfico, las rutinas de atención de interrupciones, etc...

El modelo S.O. tiene una entrada para el ingreso de trabajos y su correspondiente salida para su finalización. Además tiene las entradas y salidas para comunicarse con los otros modelos, como el establecimiento del tiempo y la recepción del aviso de tiempo cumplido (ambos para la comunicación con el Reloj). Tendrá las salidas "ejecutar" y " tiempo" para enviarle mensajes a la CPU y las entradas correspondientes a "Fin", "E/S" y " desalojo" para recibir mensajes de la CPU (la funcionalidad de estos mensajes está comentada en la descripción del modelo CPU).

El AdministradorTrabajos recibe un trabajo, genera internamente un proceso para dicho trabajo, y lo envía al Coordinador. Para esto tiene una entrada y una salida correspondientes.
A su vez, cuando finaliza un proceso, el camino se invierte: el Coordinador le envía el proceso

finalizado, el AdministradorTrabajos debe “ubicar” el trabajo correspondiente e indicar su finalización. Para estos hechos, también hay una entrada y una salida correspondientes.

El modelo BloqueadoE/S representa a la cola de bloqueados esperando la finalización de una Entrada/Salida. Requiere sólo la entrada de los procesos y la salida de los mismos. Además, se reutilizará el modelo Reloj para simular la entrada/salida física. Es decir, el proceso estará bloqueado un cierto período de tiempo y una vez cumplido, el reloj generará un evento que provocará la salida de la cola del primer proceso bloqueado.

El modelo Listos es muy similar (también ingresan y egresan los distintos procesos) pero con la diferencia que la salida de los mismos se realiza como respuesta a un pedido del Coordinador, por lo tanto se requiere una nueva entrada para este hecho.

El modelo Controlador principalmente es un “pasador de mensajes” con un mínimo de lógica (ya que distintas entradas pueden determinar una misma salida). En el marco de este trabajo, se asumirá que es este Controlador quien mantiene el “quantum” que el sistema le asignará a los procesos.

En síntesis, el funcionamiento normal de este sistema operativo es el siguiente:

- Los nuevos trabajos ingresados se encolan en la cola de *Listos*
- El sistema toma un proceso de la cola de *Listos* y lo coloca en la *CPU* para que procese
- El sistema establece un *quantum* (intervalo de tiempo) para que el proceso ejecute
- Si agota el *quantum*, el proceso es *desalojado* y se lo envía nuevamente a la cola de *Listos*
- Si el proceso indica su *finalización*, se le informa al *Administrador de Trabajos* y sale del sistema.
- Si el proceso pide una *Entrada/Salida*, se lo envía a la cola de *Bloqueados por E/S*
- Una vez finalizada la *Entrada/Salida* se lo ubica nuevamente en la cola de *Listos*
- Siempre que la *CPU* está libre, debe tomar un proceso de la cola de *Listos* para ejecutar (si que hay alguno)

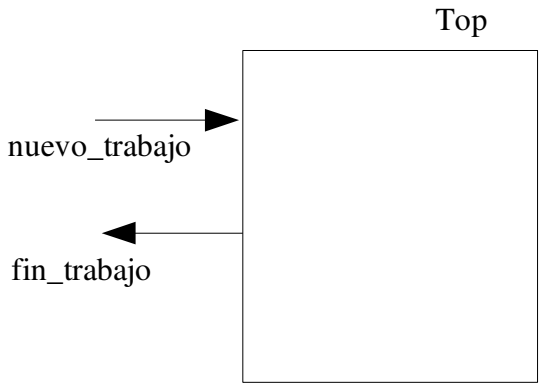
Modelo Conceptual

En esta sección, se comentarán y especificarán los modelos partiendo del primer nivel (top) y descendiendo por niveles hasta llegar a los modelos atómicos. Al llegar a uno de estos modelos, se dará la especificación del mismo. Una vez finalizados estos, se especificarán los modelos acoplados.

Por último, se propondrá la estrategia de testing para cada modelo.

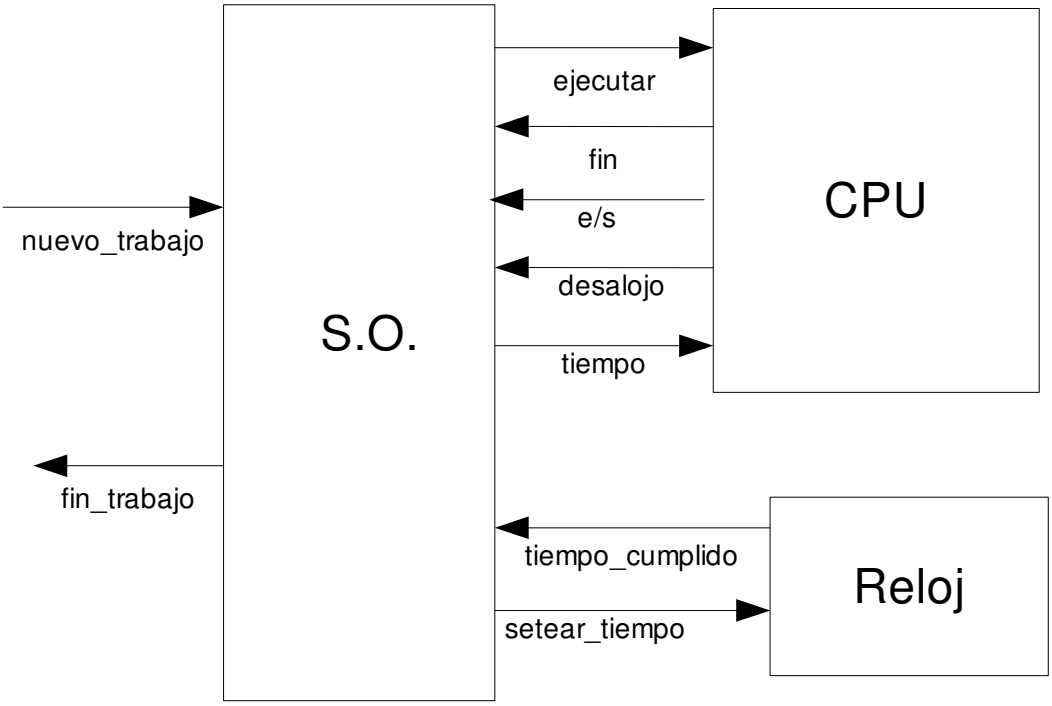
Primer nivel (top)

El nivel superior del modelo es muy sencillo ya que tiene muy poca interacción con el “mundo exterior”. Sólo tiene una entrada “nuevo_trabajo” y una salida “fin_trabajo” para el ingreso de un trabajo a procesar y la salida cuando éste haya finalizado, respectivamente.



Segundo nivel

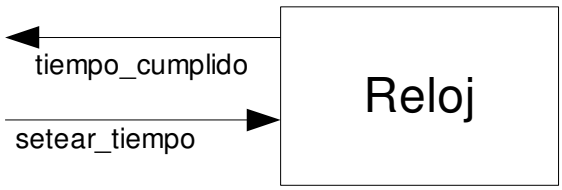
Este nivel es la explosión del modelo acoplado Top. Aquí aparecen el Sistema Operativo (S.O.) y los elementos de hardware CPU y Reloj.
Los componentes de hardware no tienen contacto entre sí. Ambos son componentes atómicos, mientras que el S.O. es un modelo acoplado.
Los puertos de entrada del modelo Top se conectan con los del mismo nombre del modelo S.O., y éste a su vez se conecta con ambos modelos "de hardware".



Modelo Atómico Reloj

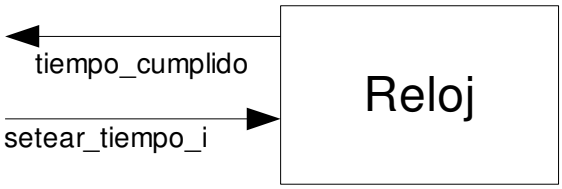
Este modelo representa al "reloj de intervalos" al que el sistema operativo le asigna una cantidad tiempo y una vez cumplida, el Reloj indica el "tiempo cumplido".

Por lo tanto, el modelo tiene una entrada ("setear_tiempo") y una salida ("tiempo_cumplido").



Formalmente, los eventos que recibe el modelo deben indicar el comportamiento que debe tomar.

Por lo tanto no puede recibir un único evento, sino uno por cada “tiempo válido”. Por ejemplo, si el reloj tiene 10 bits para almacenar este valor, los eventos serán setear_tiempo_1, setear_tiempo_2, ..., setear_tiempo_1023.
En este trabajo se generalizará como setear_tiempo_i, donde i es un natural que representa a la cantidad de milisegundos (elegido arbitrariamente, podría ser otra unidad de tiempo, pero debe ser fija) .
Este problema no aparece en la salida, ya que sólo indica que se cumplió el tiempo, independientemente de cuánto haya sido.
A continuación se muestran el diagrama del modelo Reloj y su especificación formal.



Reloj = < S, X, Y, δ_{int}, δ_{ext}, λ, ta >

S = N x {*activo, pasivo*}

N indica un número natural y representa la cantidad de milisegundos de espera.
En lo que resta del trabajo, se sobreentenderá que N indica un natural. Sólo se indicará a que se está representando con ese número.
{*activo, pasivo*} representa al estado del reloj, es decir si está seteado y *activo,* o si aun no fue seteado y está *pasivo.*

X = N (*setear_tiempo_i)*

N representa la cantidad de milisegundos de espera. De esta manera se logra tener los distintos eventos de entrada setear_tiempo_i.

Y = *tiempo_cumplido*

El modelo tiene un único puerto de salida

δ_{int} (n, *activo*) = (INF, *pasivo*)

Si el reloj está activo y se produce una transición interna, esto indica que se cumplió el tiempo de espera, por lo que se generará la salida (se verá luego en la función λ) y se pondrá al reloj en estado pasivo.

No hay transiciones internas si está en modo pasivo ya que indica que el reloj no fue seteado.

A partir de aquí los casos que no tienen sentido no se especificarán y se comentarán sólo los que tengan relevancia.

δ_{ext} (n, *pasivo*, E, X)= (n, *activo*)

Al no estar especificada esta función cuando el estado es *activo*, no se permite que lleguen eventos mientras el reloj está en ese estado.
De aquí en adelante sólo se especifican los valores válidos para esta función.

λ (n, *activo*) = *tiempo_cumplido*

Como se indicó en δ_{int} al cumplirse el tiempo se genera la salida.

ta (n, *activo*) = n

ta (n, *pasivo*) = INF

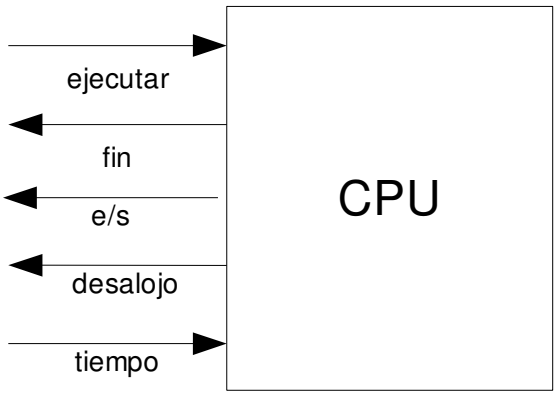
El tiempo para la próxima transición interna al setear el reloj (ponerlo en modo *activo*) es *n*, es decir, la cantidad de milisegundos que llegaron como evento de entrada.
Al finalizar el tiempo, se pasa a modo pasivo y sólo puede ser cambiado de estado por un

evento externo.

Modelo Atómico CPU

Este modelo representa al procesador y su funcionamiento es el siguiente:

1. Ingresa un proceso a “ejecutar”
2. El procesador evalúa que tiene que hacer el proceso y de acuerdo a eso
    1. genera una salida “fin” si el proceso termina
    2. genera una salida “e/s” si el proceso tiene que hacer una Entrada/Salida
    3. se mantiene en la CPU si el proceso debe procesar
3. En este último caso (2.3) se puede recibir un evento “tiempo” por lo que el proceso debe ser desalojado, lo que se informa con la salida “desalojo”.



En los tres casos de salidas y en el caso de la entrada “ ejecutar” se da la misma situación que en la entrada del Reloj. Allí se debía indicar un *tiempo* y en este caso se debe indicar un *proceso*. Por que se repetirá la estrategia de tener una entrada por cada valor posible. Para lograr esto, se asumirá que los procesos se representan con un número natural.

El número que representa a un proceso mantiene información del mismo. Los números serán de 5 dígitos que se descomponen de la siguiente manera:

$\underline{d_1\, d_2}\ \ d_3\ \ \underline{d_4\, d_5}$

id e/s procesamiento

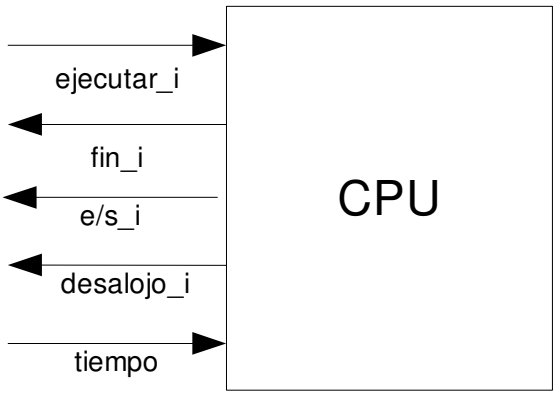
Es decir, los dos primero dígitos identifican al proceso y se mantienen invariables.

El tercer dígito indica la cantidad de entradas/salidas que efectuará el proceso. Inicialmente iba a ser el porcentaje de E/S (sin indicar cuándo se realizaban), pero se prefirió hacer el modelo determinístico para poder analizar los resultados.

Los últimos dos dígitos indican la cantidad de veces que debe ejecutar ese proceso (de acuerdo al modelo elegido, indica cuántas veces agotará el *quantum* asignado, no se incluye en este número las entradas/salidas).

A medida que avanza el sistema, se irán decrementando los valores de entrada/salida y de procesamiento acorde se vayan realizando. Cuando ambos valores lleguen a cero, en su próxima ejecución, el proceso terminará. Se priorizará el valor de las E/S (compensando el dígito menos) de la siguiente forma: si ambos valores son distintos de cero, se multiplica la cantidad de E/S restantes por 10 antes de compararlas con el procesamiento restante.

Finalmente, el diagrama del modelo CPU y su especificación formal son los siguientes:



CPU = < S, X, Y, δ_{int}, δ_{ext}, λ, ta >

S = {*procesando, recibiendo_proceso, haciendo_e/s, finalizando, desalojando, ocioso*}
X = N U{tiempo} (*ejecutar_i* U {*tiempo*})
N representa a las entradas *ejecutar_i.*
Y = (*fin_i* U *e/s_i* U *desalojo_i*)

δ_{int} (*recibiendo_proceso*) = *evaluarProceso(i)*
δ_{int} (*haciendo_e/s*) = *ocioso*
δ_{int} (*finalizando*) = *ocioso*
δ_{int} (*desalojando*) = *ocioso*

evaluarProceso(i) = Si (E/S_restantes==0 y procesamiento_restante==0)
finalizando
Si (E/S_restantes!=0 y procesamiento_restante==0)
haciendo_e/s
Si (E/S_restantes==0 y procesamiento_restante!=0)
procesando
Si (E/S_restantes!=0 y procesamiento_restante!=0)
*Si (E/S_restantes * 10 > procesamiento_restante)*
haciendo_e/s
Si no
procesando
(donde i es el único proceso que está dentro de la CPU)

δ_{ext} (*ocioso*, E, *ejecutar_i*)= *recibiendo_proceso*
δ_{ext} *(procesando, E, tiempo)= desalojando*

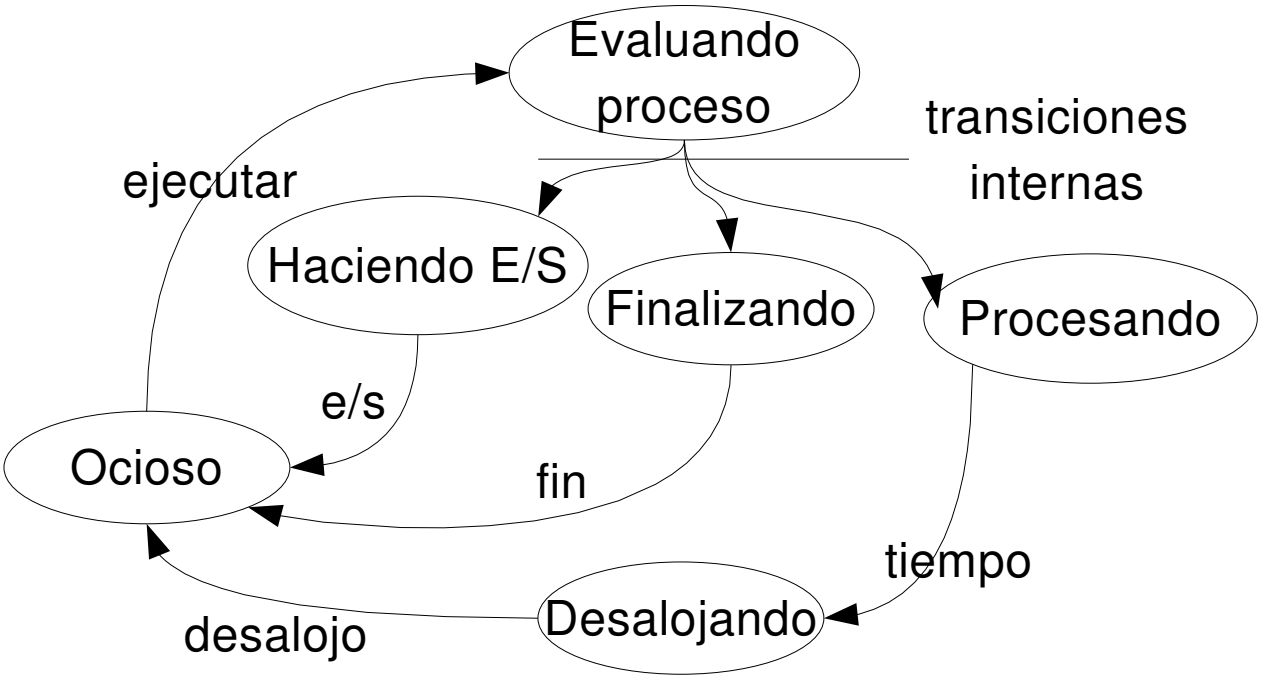
λ (*desalojando*) = *desalojo_i*
(donde i es el único proceso que está dentro de la CPU)
λ (*haciendo_e/s*) = *e/s_i*
λ *(finalizando) = fin_i*

ta (*procesando*) = *INF*

ta (*recibiendo_proceso*) = 0
ta (*haciendo_e/s*) = 0
ta (*finalizando*) = 0
ta (*desalojando*) = 0
ta (*ocioso*) = INF

Cabe destacar que una vez que un proceso toma el procesador (estado *procesando*) no lo abandona voluntariamente. Si no que debe ser desalojado.

El diagrama de estados de este modelo sería el siguiente:



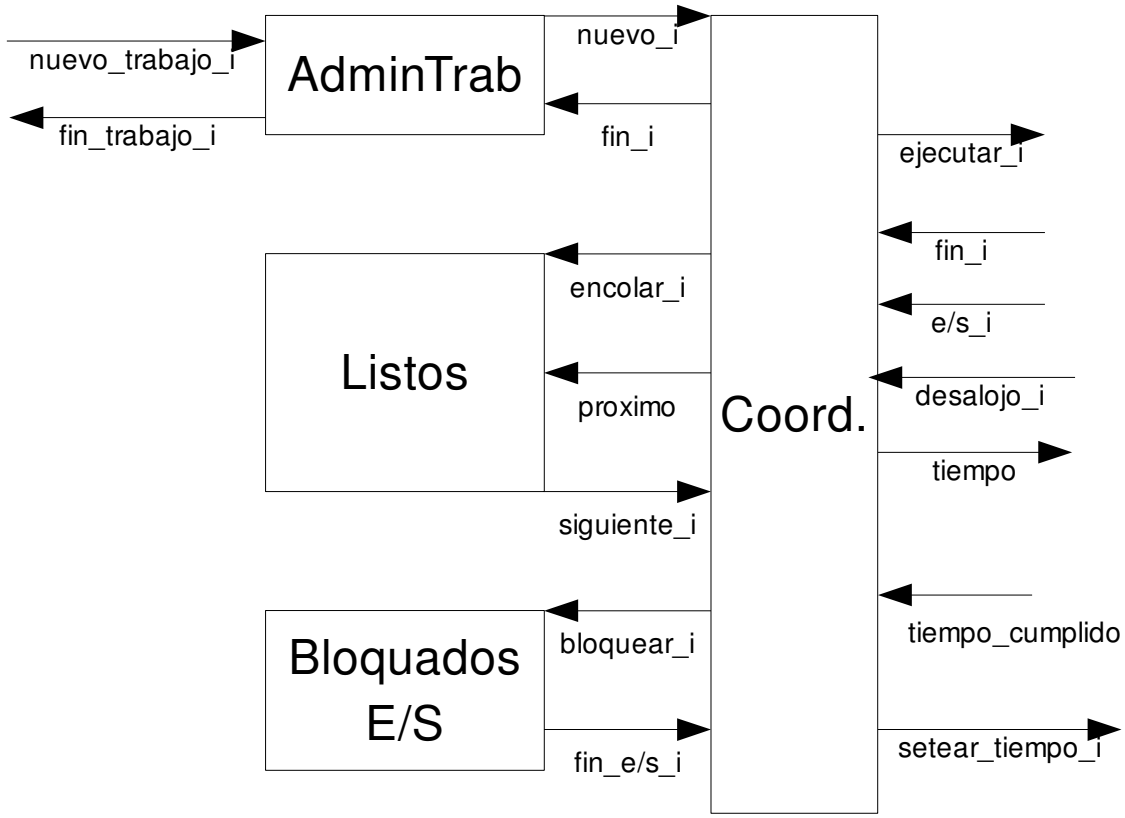
Tercer nivel - Modelo S.O.

Este es un modelo acoplado que representa al Sistema Operativo. Se descompone como se ve en su diagrama, en cuatro modelos: AdministradorTrabajos, Listos, BloqueadosE/S y el Coordinador.

En este nivel aparece la situación que surgió con el Reloj y se repitió a lo largo de los modelos, que los eventos deben representar a procesos, por que no es suficiente un único evento (que indica la funcionalidad) sino que son necesarios uno por cada evento posible (generalizado por el sufijo _i).

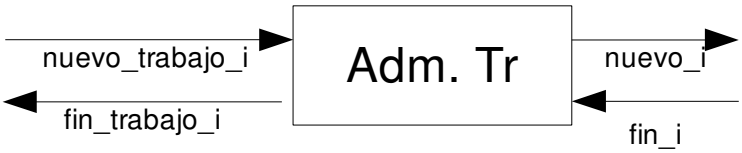
Esto introduce modificaciones en los diagramas de los modelos acoplados superiores (S.O. y Top) pero se preferió dejarlos para una mayor claridad visual. Al final de los modelos se presentará un diagrama general con los eventos correctos.

Asimismo, esto se aplica a todos los sub-modelos derivados, por lo que se usaron directamente los eventos con el sufijo _i (por supuesto, para los que así lo requieran).



Modelo Atómico AdministradorTrabajos

Este modelo representa, como su nombre lo indica, al Administrador de Trabajos. Este se encarga de recibir los trabajos del exterior, generar el proceso e ingresarlo al sistema.



AdministradorTrabajos = < S, X, Y, δ_{int}, δ_{ext}, λ, ta >

S = {*esperando, entrando_trabajo, finalizando_trabajo*}
X = (*nuevo_trabajo_i* U *fin_i*)
Representa a las entradas *nuevo_trabajo_i* y *fin_i.*
Y = (*nuevo_i* U *fin_trabajo _i*)
Representa a las entradas nuevo_i y *fin_trabajo_i.*
δ_{int} (S) = *esperando*
(δ_{int} (espera) no ocurre nunca)

δ_{ext} (*esperando*, E, *nuevo_trabajo_i*)= *entrando_trabajo*
δ_{ext} *(esperando, E, fin_i)=finalizando_trabajo*

λ (*entrando_trabajo*) = *nuevo_i*
donde i es crearProceso(j) y j es el único trabajo que está dentro del AdministradorTrabajos
λ (*finalizando_trabajo*) = *fin_trabajo _i*

crearProceso(i) = j
Por simplicidad y ya que no aporta al objetivo del trabajo, se tomarán los procesos iguales a los trabajos ya que ambos se representan con enteros.

ta (*entrando_trabajo*) = 0
ta (*finalizando_trabajo*) = 0
ta (*esperando*) = INF

Modelo Atómico Listos

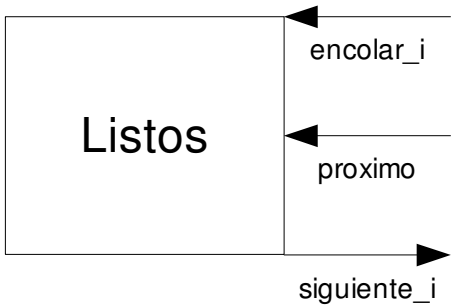
Este modelo representa, como su nombre lo indica, a la cola de Listos. Los procesos ingresan por distintas razones:

- Procesos nuevos
- Procesos que terminaron una Entrada/Salida
- Procesos que estaban ejecutando y fueron desalojados

Independientemente de la causa, ingresan por el mismo puerto. Es decir, el proceso *i*, ingresará en cualquiera de los tres casos por el puerto *encolar_i*. A partir de ese momento, el origen de su ingreso es indistinguible (ya que no es necesario para el problema) .

Cuando es solicitado un proceso (mediante la entrada "próximo") se ejecuta el algoritmo de planificación que determinará cuál de todos los procesos que se encuentran en la cola debe ser elegido para ejecutar.

Una vez elegido, se enviará dicho proceso por el puerto de salida *siguiente_i.*



Listos = < S, X, Y, δ_{int}, δ_{ext}, λ, ta >

S = <*i x* N^i> x {*esperando, desencolando*}
donde i es un natural que indica la cantidad de procesos que hay en la cola y N^i son cada uno de esos procesos (como se indicó previamente, los procesos se representan con naturales).

X = N U {*próximo*} (*encolar_i* U {*próximo*})
el natural representa a las entradas *encolar_i.*

Y = N (*siguiente_i*)
Representa a las entradas *siguiente_i.*

δ_{int} (<*i,L*>, *desencolando*) = <*i-1,quitar(L.planificador(L))*>, *esperando*
Por supuesto *i* debe ser mayor que 0 (tiene que haber algún proceso en la cola)

δ_{ext} (<n,*L*>, *esperando*, E, encolar_*i*)= <*n+1,agregar(L.i)*>, *esperando*
δ_{ext} (<*n,L*>, *esperando, E, próximo)*= <*n,L*>, *desencolando*
Si se está *"desencolando"* no se reciben nuevos eventos.

L = N^i *es decir es la lista de procesos* <I_1,I_2,...,I_n>.
agregar(L,i) = agregar(<I_1,I_2,...,I_n >, *i)*= <I_1,I_2,...,I_n,*i*>
quitar(L,i) = quitar(<I_1,I_2,...,I_n >, *i)*= <I_1,I_2,...,I_n,*i*> \ *i*

λ (*desencolando*) = *siguiente_i*

donde *i* es el proceso elegido por el planificador.

ta (*desencolando*) = 0
ta (*esperando*) = INF

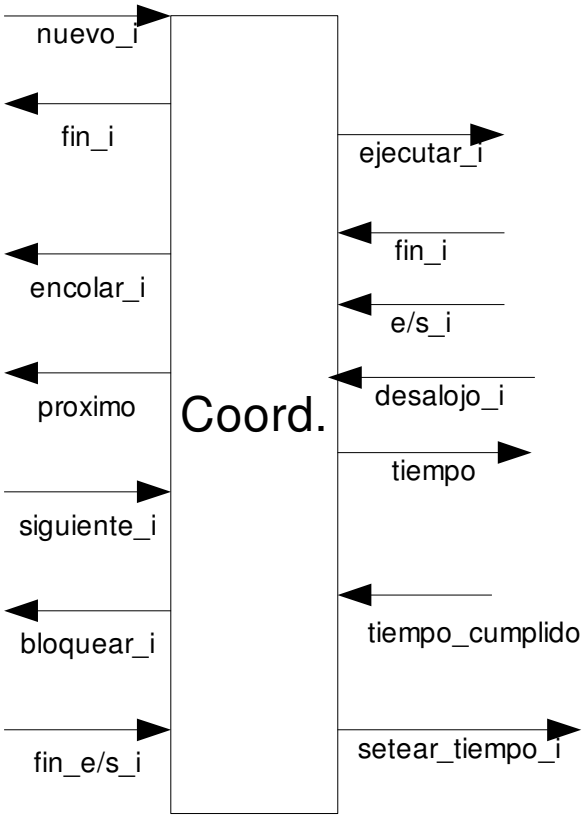
planificador es algoritmo de planificación. En este trabajo se implementaros 3 algoritmos:

- FIFO (first in-first out) : el primero que entró es el primero que sale.
  FIFO(<I_1,I_2,...,I_n >)= I_1
- LIFO (last in-first out): el último que entró es el primero que sale.
  LIFO(<I_1,I_2,...,I_n >)= I_n
- MCP (más corto primero): se elige al proceso que le resta menor cantidad de procesamiento.
  MCP(<I_1,I_2,...,I_n >)= I_j
  tal que procesamiento_restante(I_j)<procesamiento_restante(I_i) para todo i≠j.

  Y procesamiento_restante(I_i) = I_i mod 100 por la representación elegida de los procesos.

Modelo Coordinador

Este modelo representa a distintos módulos del Sistema Operativo y es el que provee la lógica del circuito Listos-Ejecutando-Bloqueado para los distintos procesos.



Coordinador= < S, X, Y, δ_{int}, δ_{ext}, λ, ta >

S = {*esperando, entrando, finalizando, encolando, pidiendo_próximo, enviando_ejecutar,*

bloqueandoE/S, desalojando, enviando_desalojo}

X = {*nuevo_i, siguiente_i, fin_e/s_i, fin_i, e/s_i, deslojo_i, tiempo_cumplido*}
Y = {*fin_proceso_i, encolar_i, próximo, bloquear_i, ejecutar_i, tiempo, setear_tiempo_i*}

[El caso *S* = *esperando* no ocurre nunca]
δ_{int} (*encolando*) = *esperando*
δ_{int} (*pidiendo_próximo*) = *esperando*
δ_{int} (*enviando_desalojo*) = *esperando*
δ_{int} (*entrando*) = decisión
δ_{int} (*finalizando*) = decisión
δ_{int} (*enviando_ejecutar*) = decisión
δ_{int} (*bloqueandoE/S*) = decisión
δ_{int} (*desalojando*) = decisión

donde *decisión* es una función que chequea si hay procesos dentro del sistema, de manera que cuando se libera el procesador (por cualquiera de las opciones posibles) decide si el *Corrdinador* debe pasar a estado *esperando* o a *pidiendo_proximo*. De esta manera garantiza que siempre que el procesador esté libre, se intentará ejecutar algún proceso. Para lograr esto, el *Coordinador* mantiene internamente la cantidad de procesos en el sistema. A su vez, esta función puede ser llamada por la entrada de un nuevo proceso, donde el coordinador, en principio, no sabe si la CPU está ejecutando o no. Por ese motivo, almacenará también cuando envía un proceso a ejecutar y cuando se libera la CPU, ya que el Coordinador es el único que tiene contacto con ella.
El pseudo-código de esta función es el siguiente:

```
Si la CPU no está ejecutando y hay procesos en el sistema
        entonces
                pidiendo_proximo
        si no
                esperando
fin si
```

δ_{ext} (*esperando*, E, *nuevo_i*) = *entrando*
δ_{ext} (*esperando*, E, *siguiente_i*) = *enviando_ejecutar*
δ_{ext} (*esperando*, E, *fin_e/s_i*) = *encolando*
δ_{ext} (*esperando*, E, *fin_i*) = *finalizando*
δ_{ext} (*esperando*, E, *e/s_i*) = *bloqueandoE/S*
δ_{ext} (*esperando*, E, *deslojo_i*) = *desalojando*
δ_{ext} (*esperando*, E, *tiempo_cumplido*) = *enviando_desalojo*

Cabe destecar que cuando llega un evento y el Coordinador está realizando una acción (es decir, no está *"esperando"*) ese evento se ignora. Asimismo, es de destacar que el estado *pidiendo_próximo* se alcanza únicamente mediante transiciones internas (desde los estados que en δ_{int} llaman a la función *decisión*.

λ *(entrando) = encolar_i*
λ *(finalizando) = fin_proceso_i*
λ (*encolando*) = *encolar_i*

λ (*pidiendo_próximo*) = *próximo*
λ (*bloqueandoE/S*) = *bloquear_i*
λ (*enviando_ejecutar*) = *{ seteando_reloj_t, ejecutar_i }*
t es el *quantum* estipulado para los procesos.
λ (*enviando_desalojo*) = *tiempo*
λ *(desalojando)* = *encolar*

ta (*S*) = 0 [si S no es *esperando*]
ta (*esperando*) = INF

En todos los casos, el *i* del sufijo debe almacenarse internamente para poder ser utilizado desde la función λ.

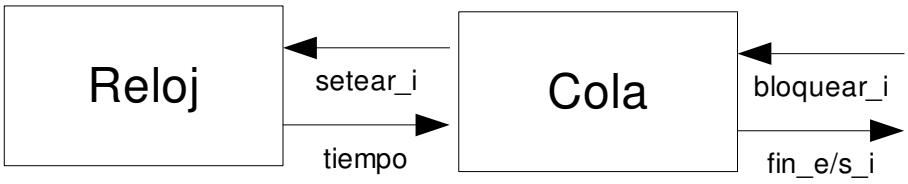
Cuarto nivel - Modelo BloqueadosE/S

Este modelo representa, como su nombre lo indica, a la cola de Bloqueados en espera de la finalización de una Entrada/Salida. Dado que el único dato que aporta al trabajo es el tiempo que un proceso está dentro de esta cola, decidí reutilizar el modelo Reloj de modo que, cuando un proceso ingresa a BloqueadosE/S, se setea un tiempo de espera que, una vez cumplido, liberará al proceso (como si hubiera terminado la operación sobre el periférico). A la vez, debe administrar un cola de los procesos que están bloqueados esperando que se libere el periférico (se supone un único periférico dedicado). Además se supone un tiempo similar de acceso para todos los pedidos de E/S.

De este modo, el modelo BloqueadosE/S es un modelo acoplado y se especificará en la próxima sección.

El modelo *Cola* es similar a Listos sólo dos diferencias: por un lado, en lugar de tener un planificador que decide qué proceso debe salir, siempre sale el primero (similar a planificador FIFO). Por otro lado, tiene un nuevo puerto de salida *setear_i.* Para desencolar recibe un evento externo igual que Listos, pero la diferencia radica en que Listos es totalmente pasivo en esta situación, mientras que esta *Cola* activa un *Reloj* (mediante el nuevo puerto de salida) para programar dicho evento. Además, si al terminar de desencolar ve que tiene más procesos, vuelve a setear el Reloj para programar un nuevo “desencole”.

En cuanto al manejo de la cola es exactamente igual que en el modelo Listos detallado dentro del nivel del Sistema Operativo (S.O.).



Modelos acoplados

Modelo Top

[top]

components : cpu@miCPU rel@Reloj SO

Out : finTrabajo

In : nuevo

% Entradas y salidas del modelo

Link : nuevo nuevoTrabajo@SO

Link : finTrabajo@SO finTrabajo

%Comunicación con la CPU

Link : ejecutar@SO ejecutar@cpu

Link : tiempo@SO tiempo@cpu

Link : fin@cpu fin@SO

Link : e_s@cpu e_s@SO

Link : desalojo@cpu desalojo@SO

%Comunicación con el Reloj

Link : setear_tiempo@SO in@rel

Link : out@rel tiempo_cumplido@SO

Modelo S.O.

[SO]

components : coord@Coordinador adm@admTr lis@Listos bloq_ES

Out : finTrabajo

Out : ejecutar

Out : tiempo

Out : setear_tiempo

In : nuevoTrabajo

In : fin

In : e_s

In : desalojo

In : tiempo_cumplido

% Entradas y salidas del modelo

Link : finTrabajo@adm finTrabajo

Link : ejecutar@coord ejecutar

Link : tiempo@coord tiempo

Link : setear_tiempo@coord setear_tiempo

Link : nuevoTrabajo nuevoTrabajo@adm

Link : fin fin@coord

Link : e_s e_s@coord

Link : desalojo desalojo@coord

Link : tiempo_cumplido tiempo_cumplido@coord

%Comunicación con el AdministradorTrabajos

Link : nuevoProceso@adm nuevo@coord

Link : fin_proceso@coord finProceso@adm

%Comunicación con la cola de Listos

Link : encolar@coord encolar@lis

Link : proximo@coord proximo@lis

Link : siguiente@lis siguiente@coord

%Comunicación con la cola de Bloqueados

Link : listo@bloq_ES fin_e_s@coord

Link : bloquear@coord encolar@bloq_ES

Modelo BloqueadosE/S

[bloq_ES]

components : bloq@Bloqueados relB@Reloj

Out : listo

In : encolar

% Entradas y salidas del modelo

Link : listo@bloq listo

Link : encolar encolar@bloq

%Comunicación con el reloj interno de Bloqueados

Link : hacerESfisica@bloq in@relB

Link : out@relB finESfisica@bloq

[bloq] //Ejemplo de inicialización del modelo Cola

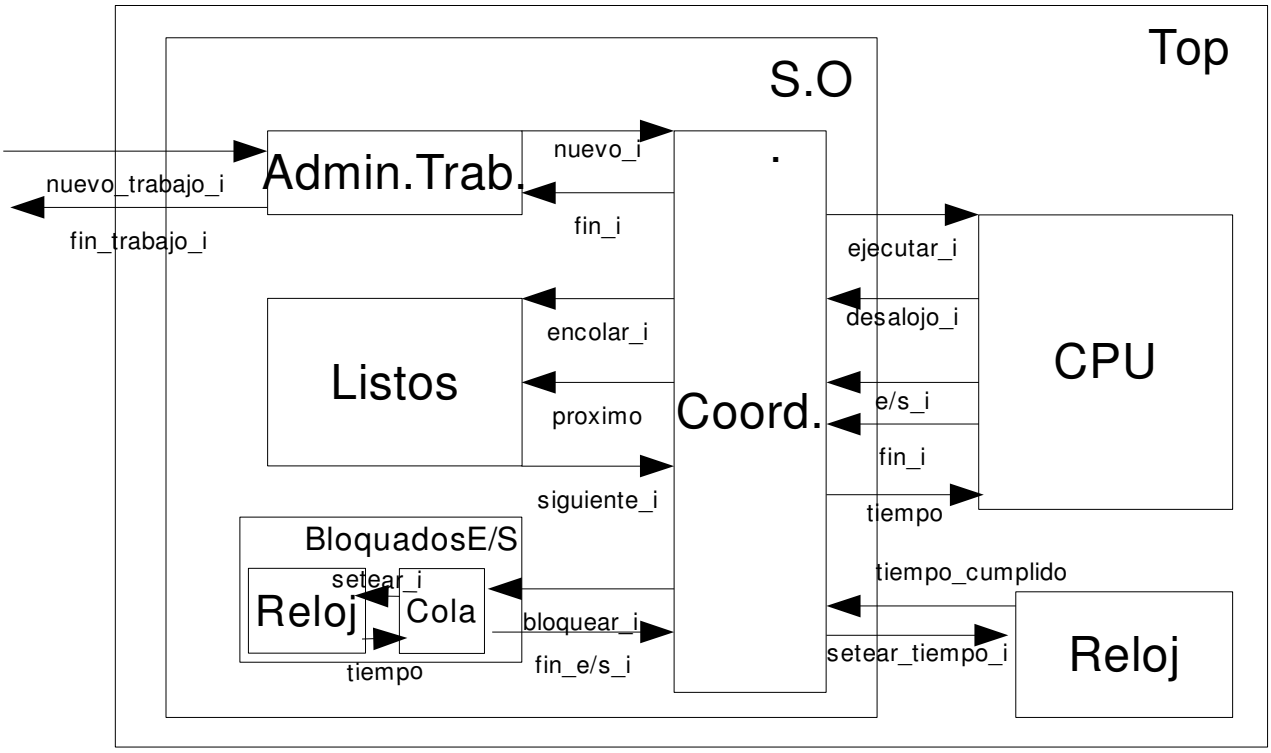
demora : 0:0:0:2 //demora es el tiempo que tarda el modelo en enviar un mensaje

Diagrama del Modelo General

La siguiente es una vista del diagrama general del modelo. Los puertos de los modelos acoplados que se conectan uno a uno con otro modelo están representados con una flecha que atraviesa los límites del modelo. Si bien esto no es real, ya que hay un puerto de entrada en un modelo que se conecta con uno de salida del otro modelo, las fechas en el gráfico dificultaban la comprensión.

Es decir, cuando una flecha atraviesa un modelo, debe entenderse que allí hay un puerto de entrada (o de salida, dependiendo del sentido de la flecha). Por ejemplo, nuevo_trabajo_i atraviesa a los modelos *top* y *S.O.* llegando a *AdministradorTrabajos.* Los tres modelos tienen un puerto de entrada que recibe dicha flecha. Asimismo, El modelo *top* "mapea" dicha entrada con el modelo *S.O.* quien a su vez lo "mapea" con el *AdministradorTrabajos.*

De la misma forma, se utilizaron los mismos nombres para los puertos que son alcanzados por dicha flecha.



Cuestiones de implementación y testing

Reloj

Para el testing de estos primeros modelos se tendrán en cuenta

1) Situaciones normales
2) Situaciones anormales (por ejemplo tiempo 0)

A medida que se avance en el trabajo, se priorizarán las situaciones normales sobre las anormales, por lo que se dedicará mayor esfuerzo a testear las primeras, de modo que se garantice el correcto funcionamiento cuando la entrada de eventos es correcta, sin anormalidades.

Ejemplo de archivo de eventos para testing del modelo:

00:00:10:00 in 10 //Caso normal

00:00:30:00 in 3 //Caso normal

00:00:49:00 in 0 //Caso anormal

00:00:50:00 in 15 //Caso normal

00:00:55:00 in 15 //Caso anormal: anula la entrada anterior

En la implementación se soluciona el inconveniente surgido en el modelo (de representar todos los valores posibles) que hizo que aparecieran muchos eventos. En el código, esto se representa por el parámetro (o *valor*) del mensaje, volviendo a tener un único puerto para representar a estos eventos.

AdministradorTrabajos

Este modelo procesa todos los mensajes en tiempo 0, por lo cual para el testing sólo hay que ver que los mensajes entrantes tengan su correspondiente salida, y que ignore los mensajes llegados cuando está activo, es decir, procesando otro mensaje. Se tuvo en cuenta que el orden de entrada y salida de los trabajos no sea el mismo.

Archivo de eventos para testing:

00:00:10:00 nuevoTrabajo 1

00:00:10:00 nuevoTrabajo 9 //Caso inválido: se ignora

00:00:30:00 nuevoTrabajo 2

00:00:50:00 nuevoTrabajo 3

00:00:55:00 finProceso 1

00:01:50:00 nuevoTrabajo 4

00:02:50:00 finProceso 3

00:03:00:00 finProceso 2 //este trabajo ingresó antes del último que salió

00:03:20:00 nuevoTrabajo 5

00:03:50:00 finProceso 4

00:04:50:00 finProceso 5

En este modelo aparecen las primeras funciones auxiliares para la transformación de *Trabajo* a *Proceso* y viceversa. Para este trabajo, ambos tipos de datos fueron definidos como enteros, y dado que el foco del trabajo está en otro lugar, no se hace ninguna transformación real de uno a otro. Sin embargo por claridad conceptual y para una posible mejora de este trabajo, se hicieron los llamados a dichas funciones. Estas funciones y definiciones de tipos fueron ubicadas en los archivos *varios.h* y *varios.cpp.*

Asimismo por mayor claridad en el testing de unidad, no se utilizó la codificación de 5 dígitos para los procesos, sino que se usaron enteros chicos ya que no se decodifica dentro de este modelo. Esta estrategia se repetirá en los modelos que no usen la decodificación.

Este modelo además tiene distintos estados. En el caso del *Reloj*, fue suficiente con los estados *activo* y *pasivo* que provee el simulador. En *AdministradorTrabajos* hay tres estados, por lo que se debió almacenar este valor en una variable. Además se debió almacenar el valor del *trabajo/proceso* que estaba transitando por el modelo, para poder utilizarlo desde la funciones de transición externa (*externalFunction*) y lambda (*outputFunction*).

CPU

El modelo CPU fue renombrado por miCPU en el código para evitar conflictos con el provisto por el CD++.

Nuevamente se almacenan el estado y el proceso que está dentro de la CPU. La función *evluarProceso* que aparece en este modelo, es propia de la CPU, por lo tanto, no se incluyó en varios, sino que quedó dentro del modelo.

Para testear este modelo, se ingresó un proceso de cada tipo y la señal de desalojo para “liberar” al que quedó procesando. A partir de éste modelo no se copiarán los archivos de testing a no ser que requieran una explicación o que aporten alguna información adicional. Estos pueden verse en los directorios correspondientes bajo el nombre *test.ev* (si es uno sólo) o *testX.ev* (donde *X* es un número).

Archivo de testing

00:00:09:00 tiempo 1

00:00:10:00 ejecutar 10001

00:00:15:00 tiempo 1

00:00:16:00 ejecutar 11101

00:00:19:00 ejecutar 12003

00:00:26:00 tiempo 1

00:00:28:00 ejecutar 13001

00:00:33:00 tiempo 1

00:00:35:00 ejecutar 140010

Resultado obtenido

00:00:15:000 desalojo 10000

00:00:16:001 e_s 11001

00:00:26:000 desalojo 12002

00:00:33:000 desalojo 13000

00:00:35:001 fin 14000

Aquí se aprecia como, además de responder adecuadamente al tipo de tarea que debe realizar cada proceso, como se va decrementando los valores en la parte de E/S y procesamiento.

Listos

Se implementa con un arreglo de procesos (*int*) de tamaño fijo (*TAM_MAX_LISTOS*).

Se crearon funciones auxiliares para mostrar el estado de la cola en un determinado momento (*mostarCola*) -principalmente para el debug del modelo-, y para eliminar un proceso (*eliminar*) de dicha cola. Como se dijo anteriormente, se programaron 3 planificaciones (FIFO, LIFO y MCP) debiendo indicarse cuál se utilizará como parámetro del modelo (en el archivo *.ma* del modelo acoplado).

El testing se realizó en forma incremental, viendo inicialmente que encolaba correctamente los procesos, luego que encolaba y devolvía en orden correcto los procesos (actualizando la cola) y por último, se agregó un caso para ver que ignoraba los mensajes que debía ignorar (*proximo* cuando la cola está vacía)

Coordinador

Para el testeo de éste modelo, dada la gran cantidad de puertos que posee, la mayor dificultad fue simular todos los modelos con los que va a interactuar. Las pruebas que se realizaron fueron, básicamente, que se pasen correctamente los mensajes y que no se "mezclen" los procesos en el medio. Para esto último se hizo que mientras un proceso estaba ejecutando, otro ingresaba. Además se invirtió el orden para la finalización, etc.

Nuevamente surgió un problema con el modelo ya que faltaba la " inicialización" de ciclo Listo-Ejecutando-Bloqueado: nunca se ejecutaba el "primer" *proximo.* De ahí surgió que tanto cuando ingresa u nuevo trabajo, como cuando finaliza una E/S y cuando un proceso abandona la CPU (por la razón que fuera) debe examinarse la cola de listos para tomar el *proximo* proceso a ejecutar.

Modelos acoplados

Dado que todos los modelos se relacionan únicamente con el coordinador, para testear la integración se probarán de a pares, el *Coordinador* y cada uno de los modelos atómicos restantes (Ej: *Coordinador-Reloj, Coordinador-CPU,* etc...). Luego se irá incrementando el acoplamiento progresivamente hasta obtener el modelo completo.

Coordinador-Reloj

Cuando se realizo el testing del Reloj, se encontró que la implementación no respetaba al modelo inicialmente diseñado, en el sentido que al llegar un mensaje y el Reloj estar activo, el modelo indicaba que debía ignorarse, mientras que el programa no lo hacía, alterando el comportamiento.. Por este motivo, se tuvo que explicitar (mediante un *if*) que cuando llega un mensaje y el Reloj está en estado activo, debe ignorarlo, cosa que en el modelo no es necesaria. Sin embargo, Al testear el Reloj junto con el Coordinador se notó que esta situación debe ser exactamente a la inversa, ya que si un proceso abandona la CPU voluntariamente (E/S o fin) el Reloj debe cancelarse. Para esto se encontraron dos alternativas: un nuevo evento de cancelación del timer, o bien, volver a la situación que anteriormente se consideraba errónea, con el consiguiente cambio en el modelo. Se decidió por esta última opción, volviendo a modificar el modelo teórico.

Coordinador-CPU

El testing de este acoplamiento fue básicamente quitar las simulaciones hechas anteriormente y observar que la ejecución de los procesos siguiera respetando el orden normal.

No se detectaron problemas en esta etapa.

Coordinador-Reloj-CPU

Ya testeados individualmente y por pares, se prueba la integración de los tres modelos. Se integran sin inconvenientes.

De la misma forma, se testearon las siguientes combinaciones *Coordinador-Listos, Coordinador-AdministradorTrabajos, Coordinador-AdministradorTrabajos-Listos, BloqueadosES (con sus submodelos) y Coordinador-BloqueadosES* que se acoplaron sin inconvenientes.

Test de integración

Por último quedaron los test de integración: los modelos S.O. y Top. El primero se acopló directamente, mientras que al testear el segundo (el modelo completo) surgió un inconveniente que no aparecía en los test anteriores: al finalizar una entrada/salida, en tiempo cero se pasaba al proceso a la cola de listos y a ejecutar. También en tiempo cero, la CPU lo volvía a bloquear por entrada/salida. De esta forma, volvía a ingresar a la cola (mediante un evento *encolar)* y, en el mismo momento, se debía " asignar el periférico" al primero que estaba en la cola. Pero el modelo no acepta dos mensajes simultáneos. Esta situación se detectó después de varias corridas y tras analizar repetidas veces el archivo de log del simulador. Es de destacar la utilidad y la claridad de dicho archivo, ya que contiene mucha información pero organizada de tal manera que es muy fácil tanto identificar mensajes específicos como seguir la secuencia de mensajes.

La solución a la situación encontrada fue hacer que la evaluación del proceso dentro de la CPU no fuera en tiempo 0 (se le asignó un segundo a esta operación). De esta manera se evita esta superposición de mensajes en la cola de bloqueados.

Resultados

Debido a las complicaciones en el desarrollo del trabajo, no se logró analizar resultados destacables. Se realizaron distintas corridas con cada algoritmo con dos combinaciones distintas de eventos (variando las cantidades de trabajos y de entrada-salida y procesamiento de cada uno) lográndose los resultados esperados. Sin embargo, si bien dichas corridas podrían confirmar la validez del modelo, no son suficientes para emitir un juicio sobre el comportamiento de los algoritmos.

Sobre la herramienta, se puede afirmar que es muy sencilla, ya que no tiene "partes compicadas" y con la documentación provista por la cátedra y el manual, se puede comenzar utilizar muy rápidamente. Merece una mención especial la facilidad de crear modelos acoplados, ya que es extremadamente sencillo definirlos y ejecutarlos.

El único inconveniente que se tuvo con la herramienta es la integración del plug-in tanto en una instalación existente de Eclipse como en una instalación nueva, siempre en la plataforma Linux.

Es de utilidad la instalación prácticamente automática que se provee para el entorno Windows.

Posibles mejoras al trabajo

Este trabajo tiene muchas cosas para mejorar, comenzando por completarlo para lograr el objetivo propuesto.

Principalmente, se puede mencionar como posibles mejoras la flexibilización de los modelos que se basan en arreglos, permitiendo pasar el tamaño máximo de los mismos como parámetros al definir los modelos acoplados. De igual manera en el quantum que se le asigna al proceso, que en este momento está fijo en el Coordinador. Esto se hizo en algunos modelos (como, por ejemplo, el algoritmo de planificación en Listos), pero por falta de tiempo no se completó en todos.

Además, se hicieron muchas simplificaciones sobre el funcionamiento de un Sistema Operativo. Se podrían agregar más módulos y rutinas, separando la funcionalidad hoy centralizada en el *coordinador,* en una cantidad mayor de rutinas más simples (por ejemplo la rutina de atención de interrupciones).

Otro aspecto a mejorar es la representación de los procesos, ya que además de brindar muy poca información de los mismos, la flexibilidad que ofrece es muy acotada.

La separación en módulos elegida permite crear diversas tablas del sistema que pueden ser agregadas sin necesidad de modificar sustancialmente el código. Por ejemplo, el administrador de trabajos podría crear el *bloque de control de proceso (BCP)* al crear un proceso, para cada nuevo trabajo, y allí almacenar información del mismo (cuestiones relativas a la entrada-salida, a la cantidad de procesamiento necesario, etc.)

Asimismo, la parte de entrada-salida está simplificada al máximo. Se podrían agregar distintos tipos de periféricos, con distintas demoras en cada uno. Se podrían simular fallas en los dispositivos (por ejemplo utilizando alguna distribución de probabilidades).

Una de las cosas destacables de esta herramienta y de la forma de modelar es que muchos de estos cambios se pueden incorporar "explotando" los modelos afectados y dejando en resto tal cual está.