Práctica de Control

SRPT, Memoria continua según necesidades y reubicable

Sistemas operativos
Grado en Ingeniería Informática
22/05/2017

Universidad de Burgos

Este trabajo se ha retro alimentado de los demás trabajos de los distintos simuladores de algoritmos de planificación de procesos, Algunos fragmentos de código del simulador pueden haber sido copiados íntegramente, o copiados y mejorados para crear el simulador, no obstante la mayor parte del script es de nueva implementación a versión 1 de este documento, para más información sobre código externo usado leer apartado 3 y bibliografía.

Agradeciemtos a todos lo que han posibilitado la realización de esta practica.

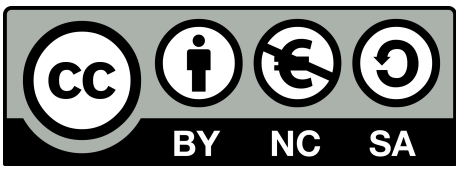
© Universidad de Burgos, España.

Escuela Politecnica superior.

Ingeniería Informática. 2016-2017.

Derechos reservados

El material presentado en este documento puede ser distribuido, copiado y exhibido por terceros siempre y cuando se haga una referencia específica a este material, y no se obtenga ningún beneficio comercial del mismo.



Cualquier material basado en este documento deberá contener la referencia *"Práctica de control SRPT, Memoria según necesidades, continua y reubicable, asignatura sistemas operativos, grado en ingeniería informática, Universidad de Burgos versión 1.0 año: 2016-2017"*.

Descripción completa de la licencia: https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es_ES

Índice de contenidos

Control de versiones

Las siguientes dos tablas indican quien o quienes han trabajado en este documento, es muy importante reconocer el trabajo hecho por otros y desde el cual partimos. También se ruega que al cierre de la practica se actualice la tabla de revisiones por parte del profesor encargado.

Tabla de versiones del documento

Versión	Alumnos	Correo electrónico	Fecha Publicación
v1.0	Santiago Hoyos Zea	shz1001@alu.ubu.es	05/2017

Tabla de revisiones del documeto

Versión revisada	Profesor encargado	Correo electrónico	Fecha de revisión
v1.0	**Pendiente de revisión**	**Pendiente de revisión**	**Pendiente de revisión**

Glosario de términos

TÉRMINO	DEFINICIÓN
SRPT	shortest remaining processing time first
Script	Archivo de órdenes, archivo de procesamiento por lotes o, cada vez más aceptado en círculos profesionales y académicos, script es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.
Bash scripting	Programación de *scripts* de *bash shell*
SFJ	Shortest Job First
CPU	Central Processing Unit (Unidad de Proceso Central)
R.R.	Algoritmo de planificación de procesos Round Robin
Clean code	Filosofía de desarrollo en el que nombres de variables, funciones y comentarios deben ser lo más explicativas posibles, sin importar el tamaño, además de otras técnicas.
MB	Mega bytes
CPU	Unidad central de procesamiento
Swapping	Técnica de mover de MP a disco duro los datos de un proceso y viceversa.
MP	Memoria principal
Unix	Sistema operativo desarrollado a principios de los 70's por Laboratorios Bell.
Unix Like	Sistemas operativos basados en la forma de trabajar de *Unix*. Ej. Linux, Mac Os X.
Array	Conjunto de valores homogéneos organizados en tablas.
Type-error	Anglicismo utilizado en ingeniería de software para referirse a errores de ortografía en literales.
Refactorización	Del inglés *refactoring* es una técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo

Introducción

Durante el transcurso del segundo semestre del primer curso del grado en ingeniería informática en la asignatura de Sistemas Operativos se han estudiado los algoritmos de planificación de procesos, la gestión de memoria, además de *bash scripting* de Linux.

Se propone una práctica de control en la que se realizará el estudio, mejora y nuevas implementaciones de ser necesario del algoritmo de planificación de procesos SRPT, una gestión de memoria continua según necesidades y reubicable. Para ello se explicará desde un punto de vista documental cada uno de los algoritmos y técnicas estudiadas en esta práctica, y luego se creará/mejorará un simulador en script de bash para Linux.

Posteriormente se realizarán dos ejercicios a modo de ejemplo para poner en práctica la teoría y probar el simulador.

Junto con este documento se podrá encontrar el script de bash para Linux (*simulador.sh*), una carpeta en un nivel inferior las versiones antiguas de este documento, además, también se adjuntan una serie de ficheros de texto plano (.txt) que pueden contener datos para las baterías de pruebas automáticas o ejemplos de salida de las mismas.

Objetivos

Estudiar, mejorar y realizar una práctica de control, para evaluar los conocimientos adquiridos en la asignatura de Sistemas Operativos del Grado en Ingeniería Informática. Se persigue un conocimiento exacto del algoritmo de planificación de procesos SRPT, de la gestión de memoria continua, según necesidades y reubicable.

Posteriormente el profesor encargado de la revisión de esta práctica evaluará el trabajo hecho y podrá exigir una defensa por parte de o de los alumnos de la misma.

1-. Shortest remaining processing time (SRPT)

SRPT es un algoritmo de planificación de procesos. Para dar una definición del mismo es necesario ver también la definición de SFJ (**S**hortest **J**ob **F**irst) y la definición de un algoritmo de planificación de procesos apropiativo y no apropiativo.

Un algoritmo de planificación de procesos apropiativo es aquel que tiene la capacidad de expulsar de CPU a un proceso cuando lo considere oportuno, basándose en una serie de criterios definidos en el. Por tanto un **algoritmo de planificación no apropiativo** es aquel que no expulsa a un proceso de CPU sino que el proceso abandona cuando ha terminado o voluntariamente.

SFJ es un algoritmo de planificación de procesos en el que entra o pasa a ejecutarse en CPU el proceso más corto o mejor dicho el que menos tiempo de CPU necesita que este en la cola en ese momento, si hay dos procesos con la misma necesidad de tiempo de procesamiento se escoge el que haya llegado primero. Si un proceso más corto del que está en proceso llega posteriormente debe esperar a que acabe el que está en CPU antes de poder pasar a ejecutarse, esta es la diferencia principal con SRPT.

Por tanto **SRPT es un algoritmo de planificación de procesos en el que siempre se está ejecutando el proceso que menos tiempo de CPU necesita**,por ende es un algoritmo apropiativo ya que expulsa de CPU y manda a la cola de listos a un proceso que se esté ejecutando si llega otro que es más corto.

Los principales problemas que surgen con este algoritmo son:

- Se ha de conocer el tiempo de CPU que va a necesitar un proceso antes.
- Los procesos más lagos pueden morir de inanición.
- El cambio de contexto puede resultar demasiado costo si se realiza entre procesos que necesitan muy poco tiempo. Ademas genera mayor sobre carga del planificador.

2-. Gestión de memoria

Otro de los temas tratados en esta práctica es la gestión de memoria, vamos a explicar brevemente desde el punto de vista documental lo que se va a realizar luego en el simulador en bash.

2.1-. Memoria Reubicable

[extracto de "Apuntes asignatura Sistemas Operativos – Tema 5, Gestión de memoria"]

El SO tiene que recalcular las direcciones sin que dependa de dónde estén localizados los programas. Por tanto traducirá las direcciones como relativas al comienzo del programa. Existen dos técnicas:

- Si la traducción de direcciones se realiza en el momento de la carga del proceso en memoria principal el proceso es **no reubicable**, se ha de recargar siempre en el mismo espacio de direcciones de MP.
- Si la traducción de direcciones se realiza el momento de ejecución el proceso es **reubicable**, se puede recargar en cualquier parte de la MP.

2.2-. Partición según necesidades

Cuando se desea cargar un proceso en memoria principal no se asignan previamente en ella particiones o porciones de un tamaño predefinido, simplemente se carga cada proceso con el espacio que se necesite.

2.3-. Memoria continua

Nos referimos con memoria continua a la técnica de mantener toda la memoria utilizada por un proceso seguida, de forma contigua. Esta puede tener el problema de que no haya suficiente espacio contiguo y se tenga que compactar el resto de memoria para que quede el suficiente **reubicando,** si aún así no se consiguiera la suficiente memoria el proceso tendría que pasar a un estado de espera hasta que haya suficiente memoria continua donde quepa.

3- Relativo al código del simulador

Simulador.sh es un script de shell bash para Linux, en el se puede encontrar todo el código necesario para la "simulación" de los conceptos vistos en el punto anterior.

El script está escrito desde 0 en la versión 1 de esta practica, no obstante haciendo buen uso de la programación modular se han aprovechado fragmentos de código de otros simuladores de otras prácticas (detalle punto siguiente).

El simulador no pretende ser un simulador real a nivel interno, solo pretende dar la sensación en la capa más alta del comportamiento, pero en código no se hace una escritura aproximada de código a como lo tendría que hacer un sistema operativo.

Todas las funciones del script tienen un pequeño comentario con el cual podemos orientarnos de que hacen. Ejemplo:

```
348  #Función que se encarga de la asignación de la memoria, reubica de ser necesario
349  function asignaMemoria() {
350
351    for ((i = 0; i < ${#nombresProcesos[@]}; i++)); do
352
353      #solo vamos a buscarles sitio a los procesos que ya hayan llegado y que no tengan
354      #memoria asignada aún y nenecisten CPU.
355      if [ ${procesosEnMemoria[$i]} -eq 0 ] && [ ${tiemposDeCpu[$i]} -gt 0 ] \
356        && [ ${tiemposDeLlegada[$i]} -le ${reloj} ]; then
357
358        #tenemos memoria para alojarlo?
359        if [[ ${memoriaNecesaria[$i]} -le $(expr $totalMemoria - $totalMemoriaOcupada) ]]; then
360
361          #recorremos toda la memoria buscando un particion donde alojar el proceso
362          inicio=0
363          contador=0
364          for ((h = $inicio; h < $totalMemoria; h++)); do
```

Es importante ver que las condiciones criticas aunque se podría intuir lo que hacen, dejamos por escrito lo que evalúan, ya que muchas veces ayuda a detectar errores y hace más fácil la lectura de la implementación.

Otro punto es la metodología de desarrollo en el script, es la llamada "clean code" en la cual no escatimamos en los nombres de las variables, solo dejamos las variables de un carácter para bucles y banderas, todo lo demás ha de ser lo más descriptivo posible sin importar la longitud (no nos cobran por espacio).

Este script ha sido desarrollado sobre el sistema operativo **Ubuntu 16.04,** con entorno de desarrollo integrado **Atom**, además se ha usado la herramienta **shfmt**(Shell Format) para la una indentación correcta y detección temprana de errores de sintaxis.

3.1-. Cambios respecto a orígenes de código

En el código del simulador de esta práctica se ha utilizado fragmentos de código del simulador de la práctica *"Práctica De control - R.R - Memoria según necesidades - continua y reubicable"* y en concreto de su archivo fuente *"RR_PD_AM_RE(modificado).sh"*.

El porqué, se basa sobre todo en que en un principio se valoró, que este código ya poseía varias de las implementaciones que se piden en el simulador de esta práctica. No obstante no se ha aprovechado todo lo que se pensaba en un principio aprovechar.

No se ha aprovechado todo lo deseado por problemas tales como una mala implementación de la programación modular, por ejemplo, Algunas funciones tan criticas como las de gestión de memoria cambian valores de variables que se desconoce su contexto o que previamente han cambiado antes en otra función y que no son variables declaradas al principio del programa o pasadas por parámetro, lo que provoca que sea bastante difícil depurar y analizar el funcionamiento del script, sobre todo a programadores novatos de shell como es el caso.

Se ha aprovechado sobre todo funciones de impresión por la salida estándar y escritura a ficheros, ya que estas eran fácilmente adaptables y funcionan bien, entre otras.

Funciones como las de asignar memoria, reubicar y el segmento de código del planificador no se han reutilizado. Las primeras por estar demasiado acopladas al funcionamiento de la lógica del planificador R.R y por tener una implementación demasiado compleja. La segunda por no ser una implementación pedida en esta práctica.

En resumen:

- **Código reescrito**
  - Bucle del planificador de procesos
  - Funciones de asignar memoria y liberar memoria
  - *Refactorización* de variables, estructura, *type-errors* etc.
- **Código reutilizado**
  - Funciones de escritura en salida estándar y ficheros (adaptadas no obstante)
  - Algunas variables
  - Lógica de planteamiento (ej. llevar los datos del programa en varios arrays ordenados por el indice)
  - Fragmento de recogida de datos: se ha llevado a una función y adaptado a la *refactorización*.

4-. Demos

Parte de esta práctica es la realización de 2 ejercicios a modo de demostración de los algoritmos y el comportamiento de la memoria, a continuación se realizarán de forma teórica sobre el papel el comportamiento deseado y luego se recreara en el simulador. Se dejarán en la carpeta de entrega los ficheros para reproducir estos ejercicios en el simulador y los informes generados.

En el simulador podremos ir viendo un resumen para cada proceso, el tiempo en el que deben llegar, el tiempo de espera acumulado, el tiempo de cpu que necesitan, la posición inicial del primer byte asignado y la posición del último.

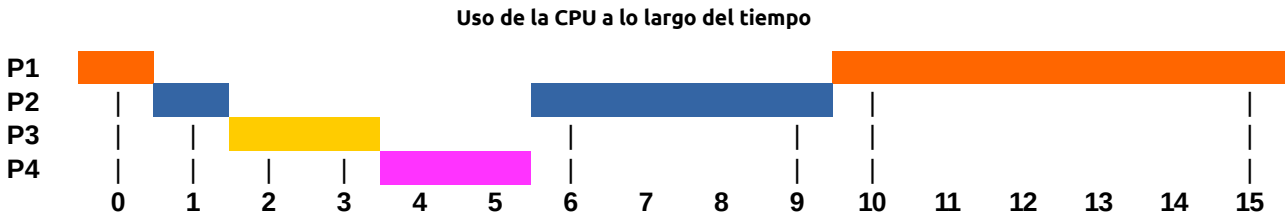
Nota importante: *posiciones y tiempos contamos siempre a partir de 0.*

4.1-. Ejercicio 1

Para esta primer ejercicio trabajaremos con **4 procesos**, donde veremos que pasa si dos procesos **llegan en el mismo tiempo**, la **elección** normal **del planificador con tiempos de ejecución distintos , y la misma con tiempos iguales**, también veremos la **asignación de memoria sin necesidad de re-ubicación**. A continuación la tabla que recoge nombre de los procesos, tiempo en el que llegan, el tiempo de cpu que necesitan consumir, y la memoria que necesitan, usaremos **12 MB de memoria del sistema**.

Nombre proceso	Tiempo de llegada	Tiempo de CPU	Memoria necesaria(MB)
P1	0	7	1
P2	1	5	2
P3	2	2	1
P4	2	2	3

4.1.1-. Demo Ejercicio 1 forma teórica

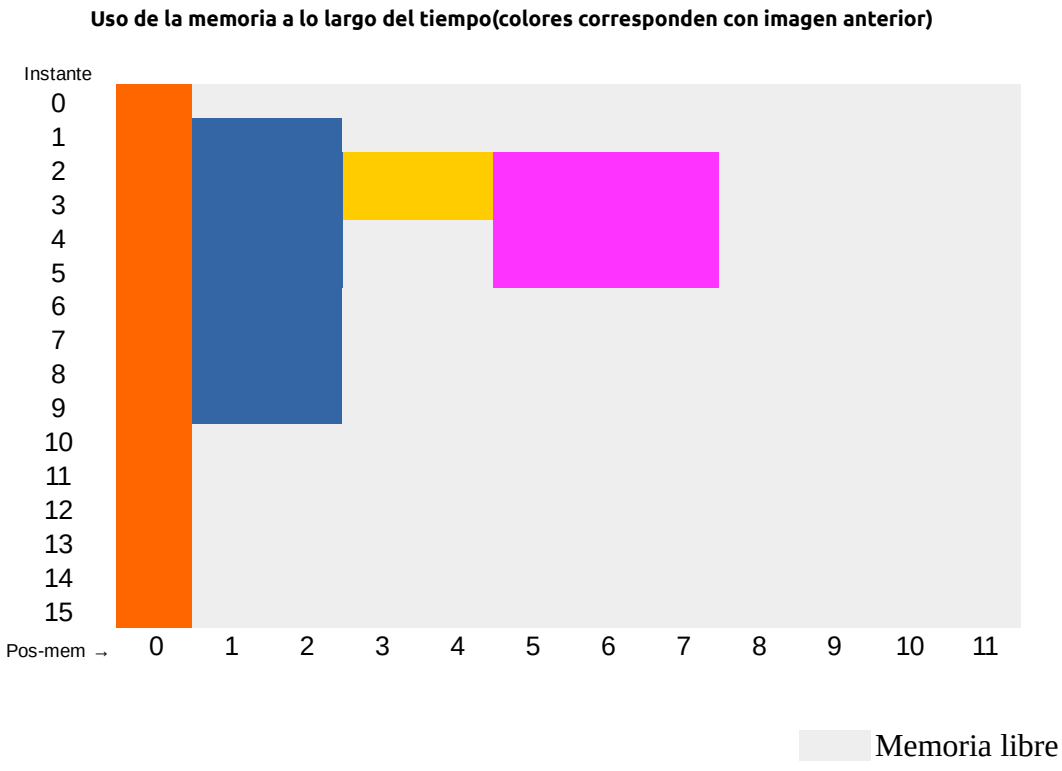


El proceso P1 llega en el instante 0 del reloj como es el único que hay actualmente se ejecuta en este tiempo, en el instante 1 llega P2, este al necesitar un tiempo de CPU más pequeño que el que esta en CPU el planificador expulsa a P1 y entra P2 a ejecutarse, P3 y P4 llegan en el instante 2 ambos necesitan el mismo tiempo, entonces, le toca elegir al

planificador al que hay llegado antes. Aunque dos procesos lleguen en el instante 0 técnicamente es imposible que lleguen exactamente en el mismo tiempo, aunque sea una diferencia de microsegundos habrá, P3 ejecuta su tiempo sin que nadie puede echarlo ya que es el que menos tiempo tiene en el instante 3 también, al acabar, entra P4. Cuando P4 acaba vuelve el planificador a meter a P2 a CPU porque necesita menso tiempo de cpu que P1, al acabar P2, P1 ya esta otra vez solo y por tanto acaba sin más.

Como podemos ver en este ejercicio P1 resulta bastante perjudicado, ya que de haberse seguido con la tendencia de que lleguen procesos más cortos que el tendría que haber seguido esperando.

La memoria no ha habido problemas, ya que ha habido suficiente para tener a todos los procesos en memoria.



4.1.2-. Demo Ejercicio 1 en el simulador

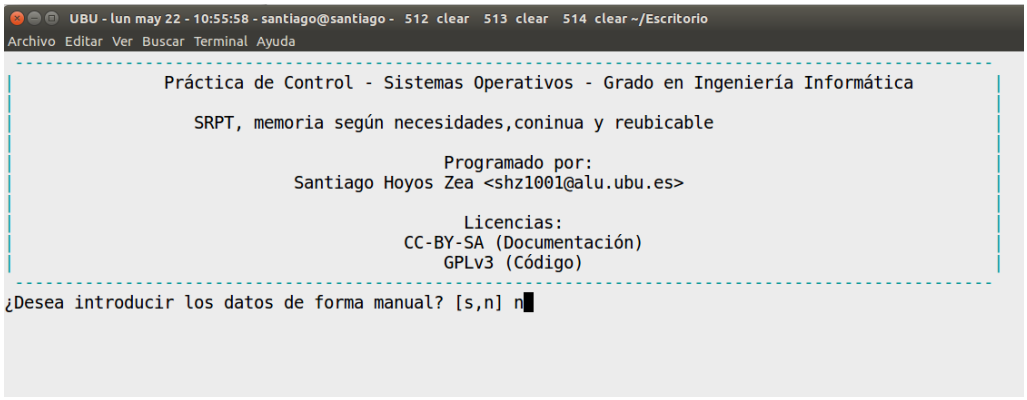
En la demo en el simulador usaremos el fichero de texto **Input.txt** con las siguientes lineas:

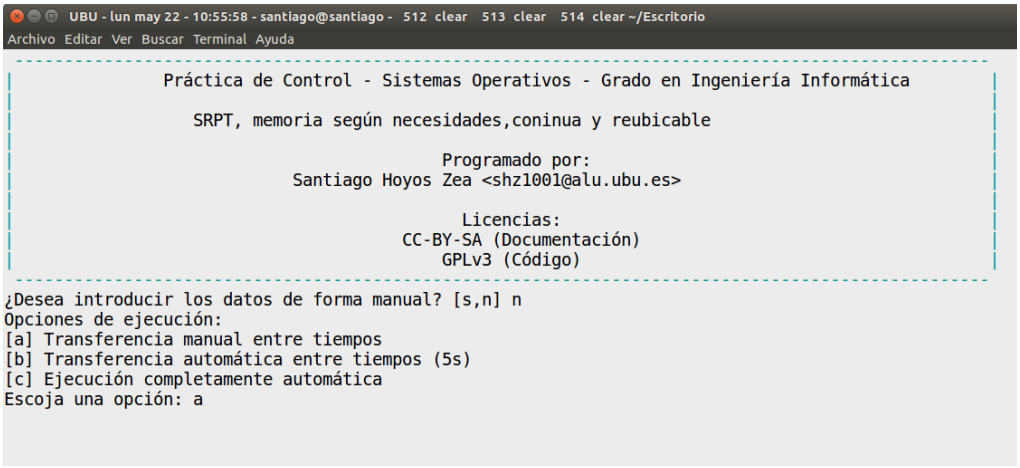
1. 12
2. p1;0;7;1
3. p2;1;5;2
4. p3;2;2;1
5. p4;2;2;3

La linea 1 indica la cantidad de memoria, las siguientes lineas son, nombre del proceso, instante de llegada, tiempo de cpu que necesita y memoria necesaria, para cada proceso.

Capturas:

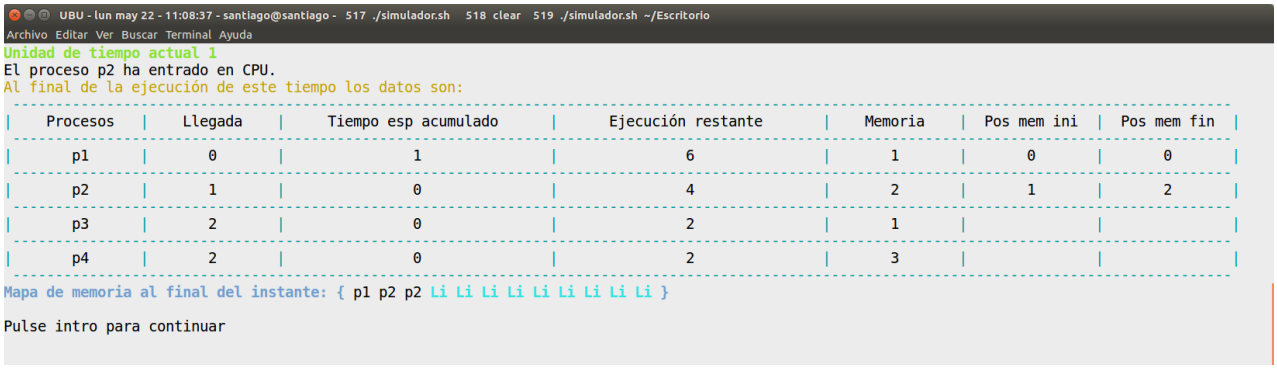
Selección de la introducción y navegación por el simulador





Instante 0, vemos como el planificado ha ejecutado P0 y en la memoria como tiene asignado su único MB necesario, los demás procesos aun no han llegado.

Instante 1, podemos ver como ha llegado P2 y al necesitar menos cpu que P1 que era el que estaba actualmente en cpu, expulsa a P1 y por tanto P1 aumenta su tiempo de espera acumulado en 1. En la memoria se le asigna a P2 de forma contigua sus 2 MB.

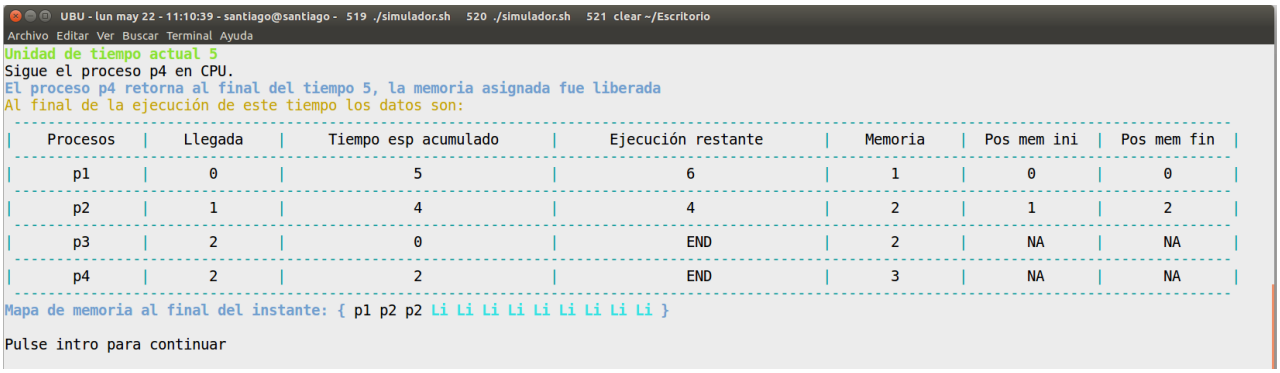


Instante 2, Llegan P3 y P4, se les carga en memoria y desde este instante pasan a ser candidatos para entrar en CPU, el planificador ejecuta P3 por tener menos tiempo que el que esta ejecutando que es P2, no elige P4 porque técnicamente ha llegado después. P1, P2 y P4 aumentan por tanto su tiempo de espera acumulado.

Instante 3, sigue P3 en CPU y acaba al final de este instante, vemos como se librea la memoria que tenia asignada. No se reubica porque no ha hace falta. Los demás procesos que no están acabados vemos como siguen aumentado su tiempo acumulado.

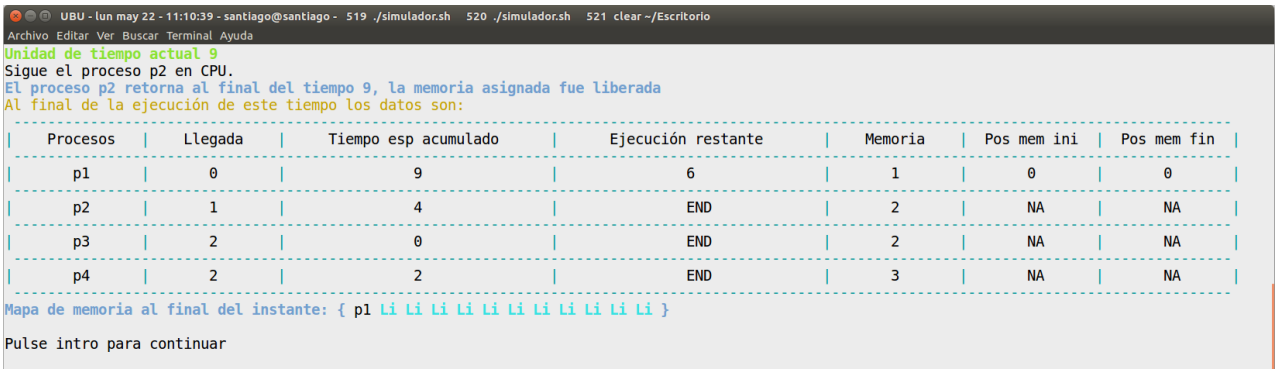
Instante 4, el planificador ejecuta P4. Los demás procesos que noe stan acabados vemos como siguen aumentado su tiempo acumulado.

Instante 5, Acaba P4 al final de este instante y se libera su memoria. Los demás procesos que no están acabados vemos como siguen aumentado su tiempo acumulado.



Instante 6, el planificado vuelve a meter a CPU al procesador al ser al que menos le falta para acabar. P1 sigue aumentando su tiempo de espera.

Instante 9, al final de este instante P2 ha acabado y se libera su memoria, P1 sigue aumentando su tiempo de espera.



Instante 10, el planificador ejecuta el único proceso que hay. Así durante los instante 11,12,13 y 14

Instante 15, el proceso P1 acaba al final de este instante, su memoria se libera.

```
UBU - lun may 22 - 11:10:39 - santiago@santiago -  519 ./simulador.sh   520 ./simulador.sh   521 clear ~/Escritorio
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Unidad de tiempo actual 15
Sigue el proceso p1 en CPU.
El proceso p1 retorna al final del tiempo 15, la memoria asignada fue liberada
Al final de la ejecución de este tiempo los datos son:
 ---------------------------------------------------------------------------------------------------------------
|    Procesos   |    Llegada    |     Tiempo esp acumulado     |      Ejecución restante      |    Memoria    | Pos mem ini | Pos mem fin |
 ---------------------------------------------------------------------------------------------------------------
|       p1      |       0       |              9               |              END             |       1       |      NA     |      NA     |
 ---------------------------------------------------------------------------------------------------------------
|       p2      |       1       |              4               |              END             |       2       |      NA     |      NA     |
 ---------------------------------------------------------------------------------------------------------------
|       p3      |       2       |              0               |              END             |       2       |      NA     |      NA     |
 ---------------------------------------------------------------------------------------------------------------
|       p4      |       2       |              2               |              END             |       3       |      NA     |      NA     |
 ---------------------------------------------------------------------------------------------------------------
Mapa de memoria al final del instante: { Li Li Li Li Li Li Li Li Li Li Li Li }
Pulsa cualquier tecla para ver resumen...
```

Resumen de tiempos acumulados y de retorno vemos lo el tiempo que ha tenido que esperar cada procesos, así como el tiempo que ha tardado en retornar.

```
UBU - lun may 22 - 11:25:50 - santiago@santiago -  520 ./simulador.sh   521 clear   522 ./simulador.sh ~/Escritorio
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
En la sigueinte tabla se ve para cada proceso el tiempo que ha pasado esperando en la cola de listos.
además del tiempo que ha tardado en retornar una respuesta.
 ---------------------------------------------------------
|    Proceso    |        Tiempo Espera Acu     | Tiempo retorno |
 ---------------------------------------------------------
|       p1      |              9               |       16       |
 ---------------------------------------------------------
|       p2      |              4               |       9        |
 ---------------------------------------------------------
|       p3      |              0               |       2        |
 ---------------------------------------------------------
|       p4      |              2               |       4        |
 ---------------------------------------------------------

santiago@santiago:~/Escritorio$
```

Se genera el informe:

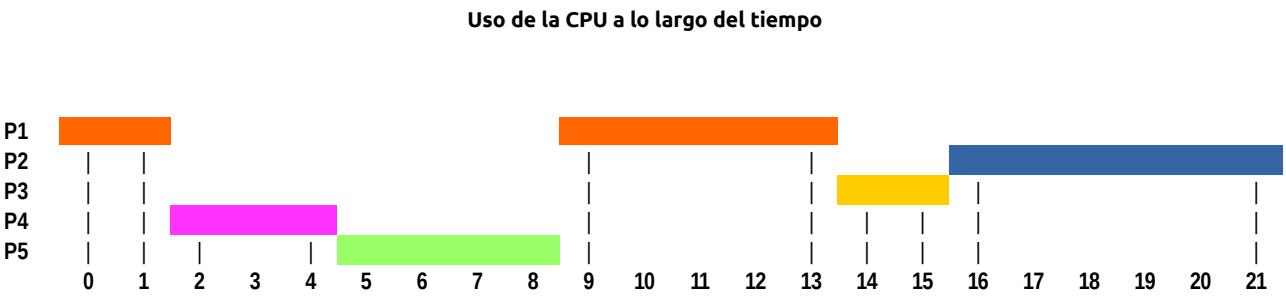
informe220
517-1110.txt

4.2-. Ejercicio 2

En este ejercicio vamos a ver dos diferencias respecto al anterior, por un lado, cómo un proceso le toca esperar a que haya memoria para poder ejecutarse aunque sea el más corto porque no se le puede alojar y cómo es necesaria la reubicación de la memoria ara poder alojar de forma contigua un proceso. Usaremos **12 MB de memoria del sistema**.

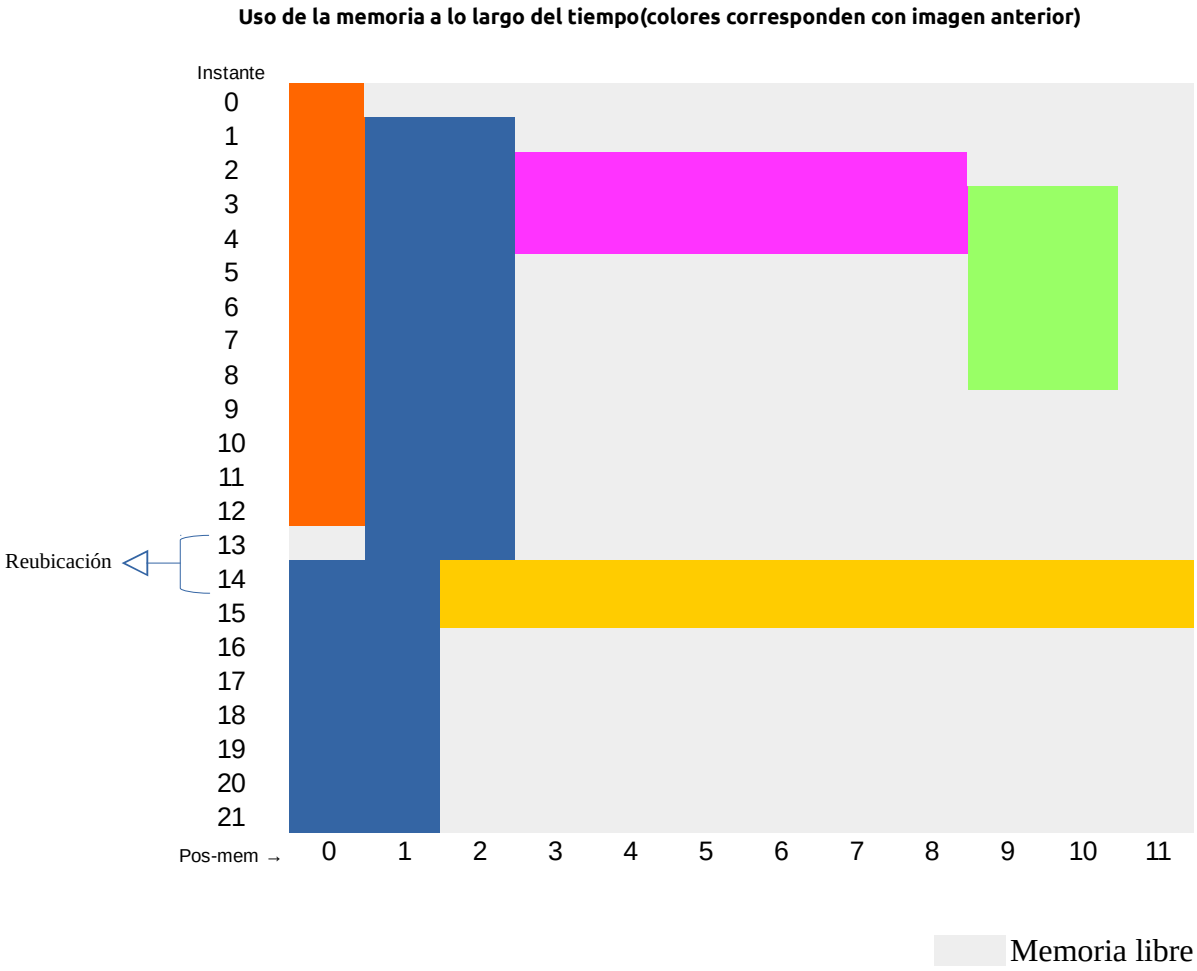
Nombre proceso	Tiempo de llegada	Tiempo de CPU	Memoria necesaria(MB)
P1	0	7	1
P2	1	6	2
P3	2	2	10
P4	2	3	6
P5	3	4	2

4.2.1-. Demo Ejercicio 2 forma teórica



El proceso P1 entra en el tiempo 0 y se ejecuta ya que es el único, posteriormente en el instante 1, llega P2 pero como P1 y P2 ahora mismo tienen una necesidad de 6 de tiempo de CPU, sigue P1 en procesador. En el instante 2 llegan P3 y P4 pero P3 aunque ha llegado primero que P4 técnicamente y tiene la misma necesidad de CPU que P4 no puede entra a CPU porque no hay memoria para alojarlo y por tanto entra a CPU P4 que si que hay memoria para el. En el instante 3 llega P5 pero como el tiempo que le falta a P4 para acabar es menor que el que P5 necesita ejecutar, se sigue ejecutando P4. En el instante 5 metemos a CPU P5 ya que ahora es de la lista, el que menos tempo de cpu necesita, al acabar P5, vuelve a entrar P1, que necesita menos tiempo de ejecución que P2, y P3 sigue sin poder entrar a ejecutarse porque no cabe en memoria, al final de P1 ya tenemos la suficiente memoria para P3 y como es más corto que P2 se ejecuta. Por último, al solo quedar P2 y tener toda la memoria para el, se ejecuta y acaba en el instante 21.

La memoria, en este ejercicio se han producido varios inconvenientes, por un lado ha habido un proceso que no se ha podido ejecutar porque no ha habido memoria para alojarle(caso de P3) y se ha producido otro problema cuando ya si que había memoria para alojarle no estaba contigua y hubo que reubicar.



4.2.2-. Demo Ejercicio 2 en el simulador

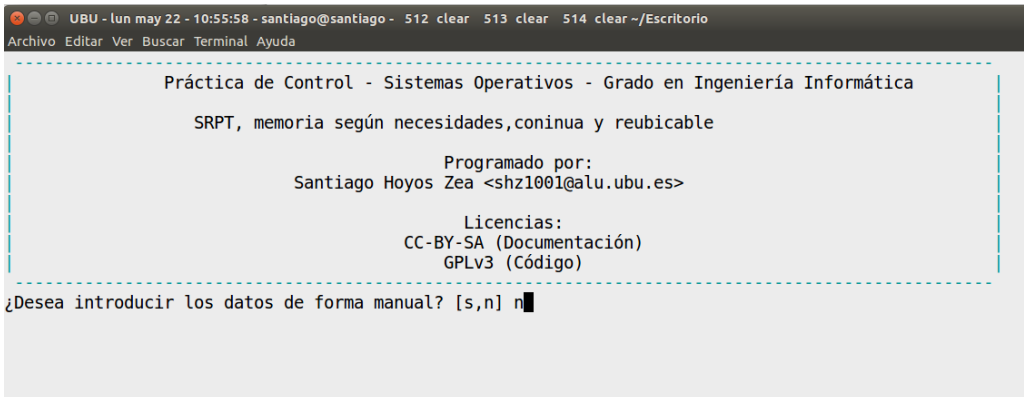
En la demo en el simulador usaremos el fichero de texto **Input.txt** con las siguientes lineas:

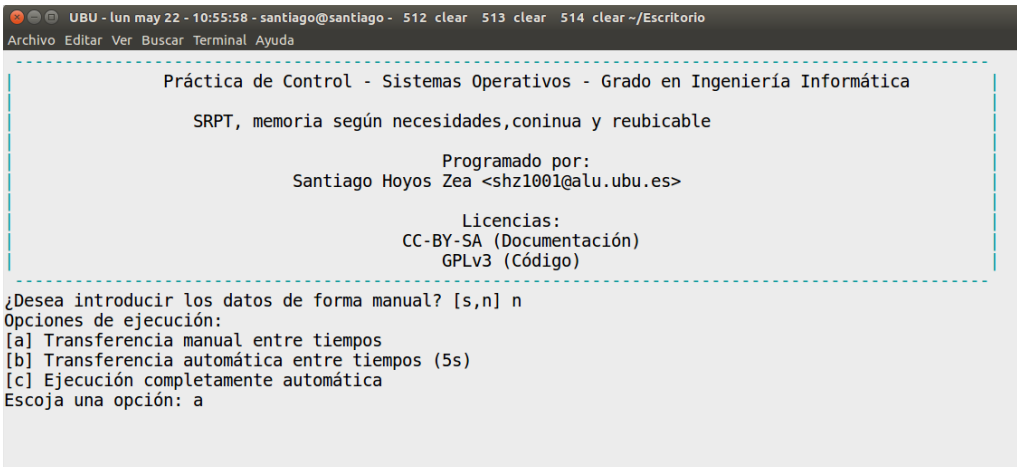
1. 12
2. p1;0;7;1
3. p2;1;6;2
4. p3;2;2;10
5. p4;2;3;6
6. p5;3;4;2

La linea 1 indica la cantidad de memoria, las siguientes lineas son, nombre del proceso, instante de llegada, tiempo de cpu que necesita y memoria necesaria, para cada proceso.

Capturas:

Selección de la introducción y navegación por el simulador

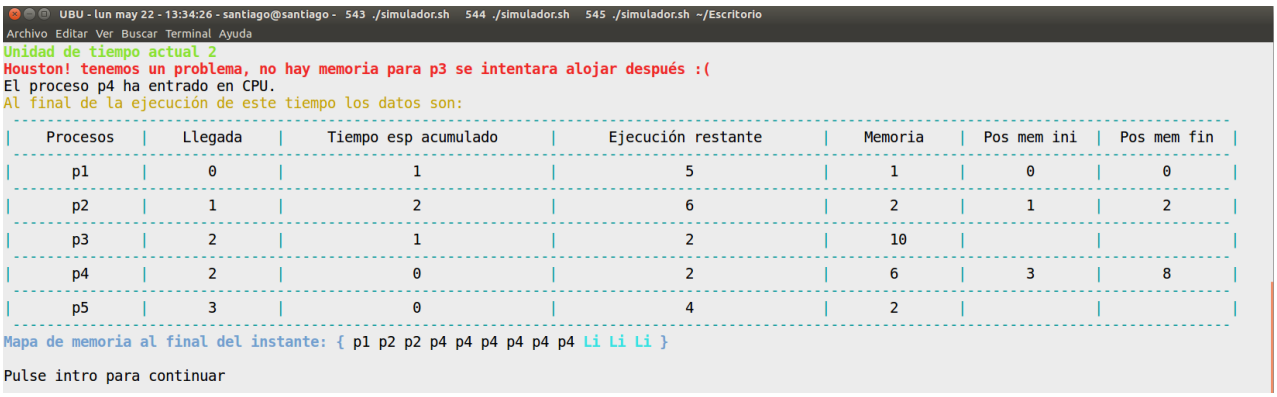




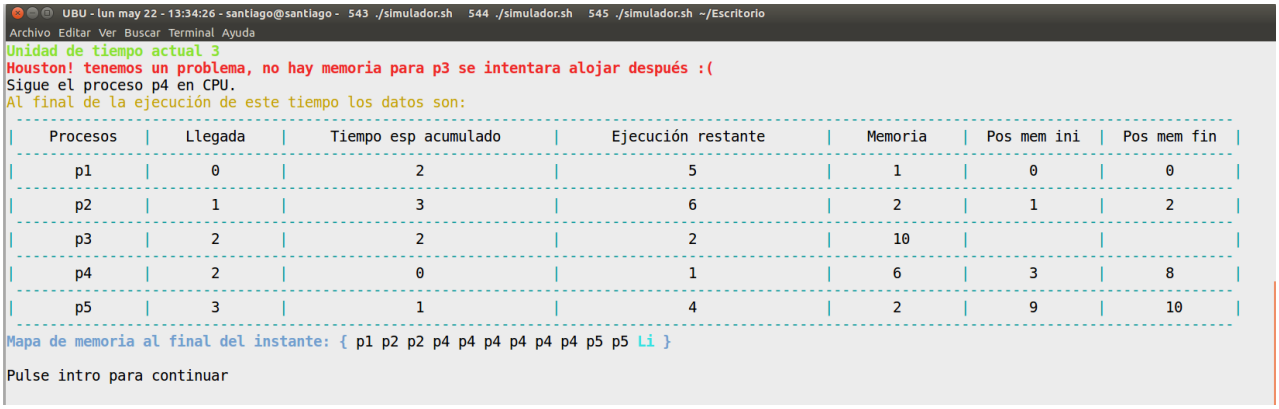
Instante 0, en el instante 0 llega a cpu el proceso p1, que entra directamente a ejecutarse por no haber ningún otro.

Instante 1, en este instante llega el proceso p2 pero al necesitar más tiempo de CPU que le proceso P1, P1 sigue en el procesador.

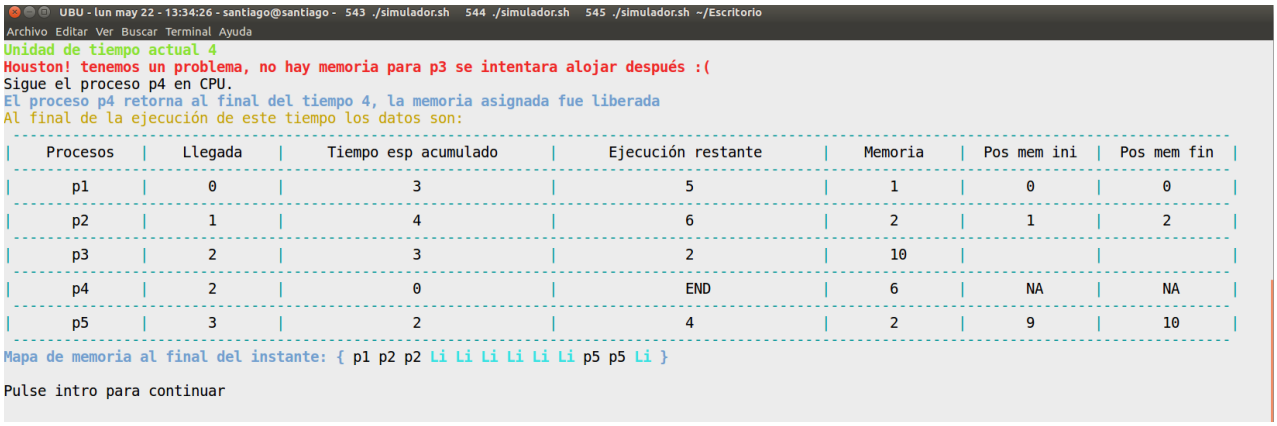
Instante 2, En este instante llegan P3 y P4 vemos como no se le podía asignar memoria a P3 ya que este exige 10 de memoria y solo hay libre (12-3)=9, no obstante el proceso P4 si que se puede alojar en memoria ya que solo pide 6 y también el planificador lo mete a cpu por ser el proceso más corto a la espera y con memoria asignada. P3 por tanto se queda a la espera de que se pueda alojar más adelante.



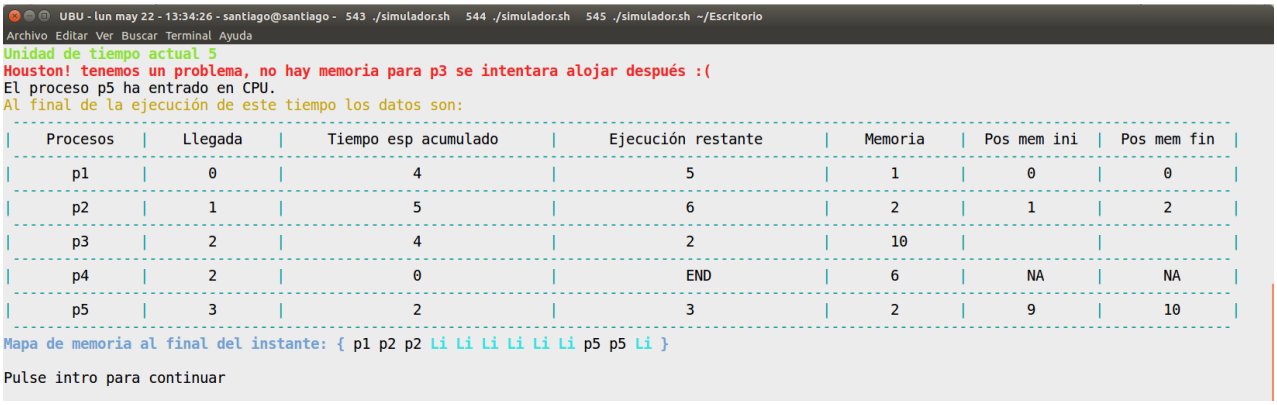
Instante 3, Llega el proceso P5 y cabe en memoria, no obstante no es mas corto que l que esta actualmente en ejecución y por tanto sigue P4 en el procesador. P3 sigue sin poderse alojar.



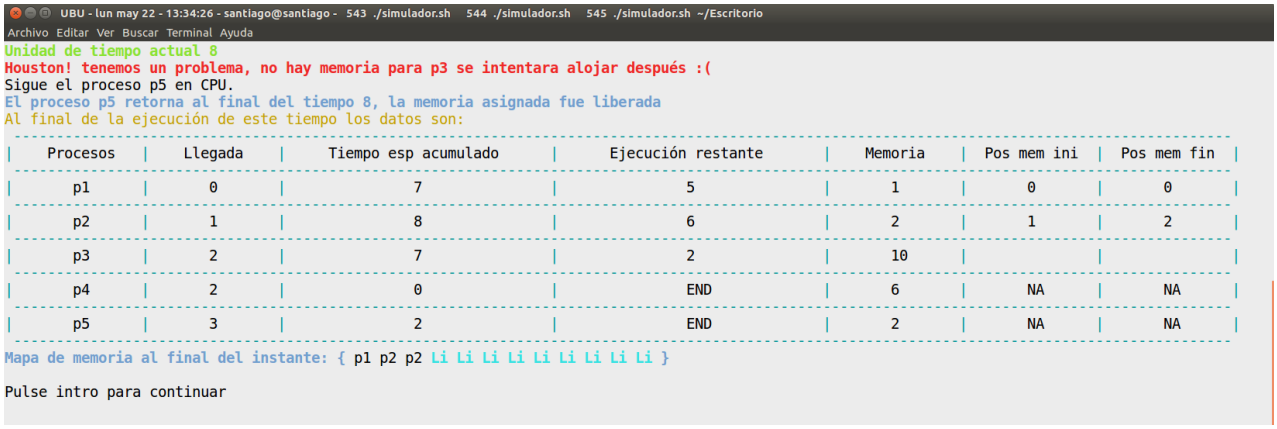
Instante 4, sigue ejecutándose P4 ya que necesita menos tiempo de procesador que el recién llegado P5. P3 sigue sin poderse alojar.



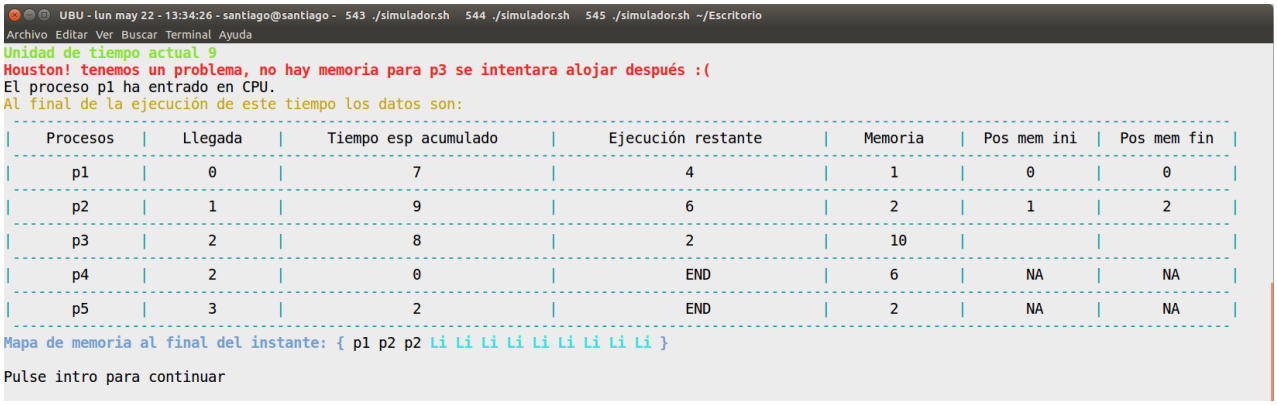
Instate 5, al terminal P4, P5 es el siguiente que menos tiempo de procesador necesita y que está en memoria principal. P3 sigue sin poderse alojar.



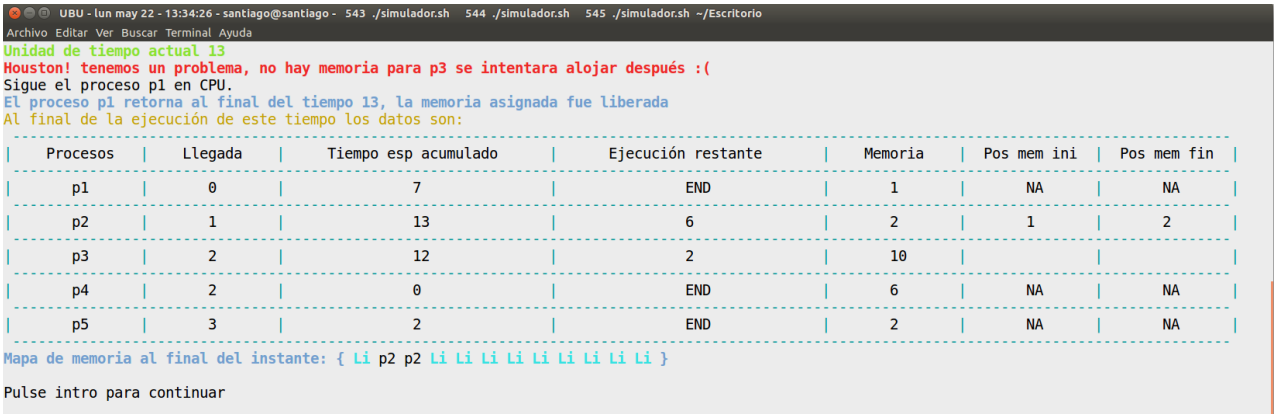
Instante 8, P5 acaba y se libera su memoria. P3 sigue sin poderse alojar.



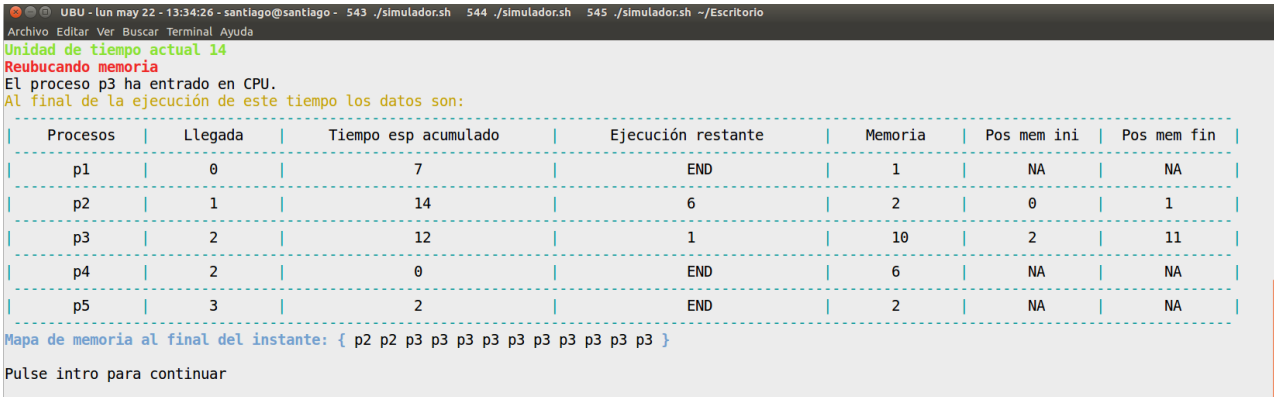
Instante 9, Como P3 sigue sin estar en memoria porque no cabe se procede a seguir ejecutando donde se dejó a P1 ya que necesita menos tiempo que P2 que es el otro proceso está en memoria y listo para ejecutarse.



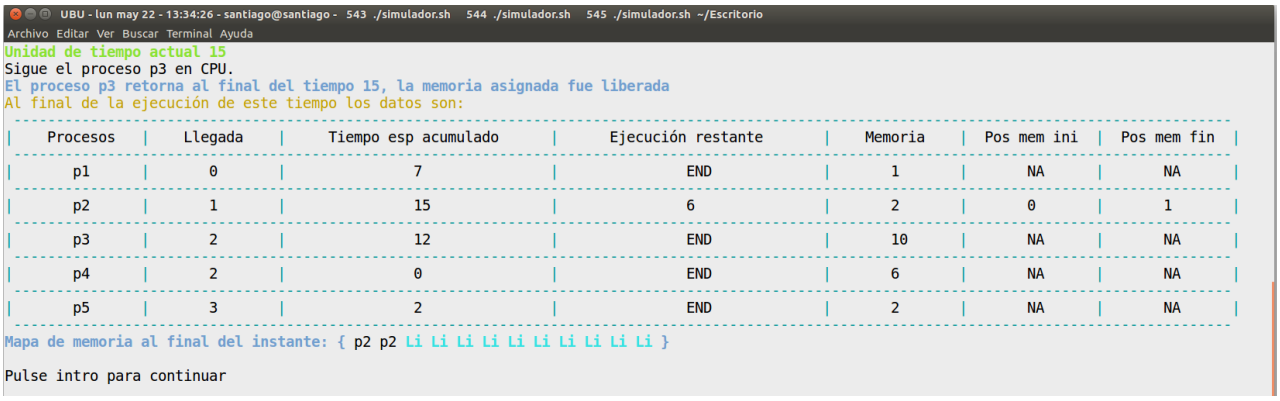
Instante 13, P1 acaba al final de esta ráfaga. Liberando su memoria, ahora tenemos 10 libre, pero no esta contigua.



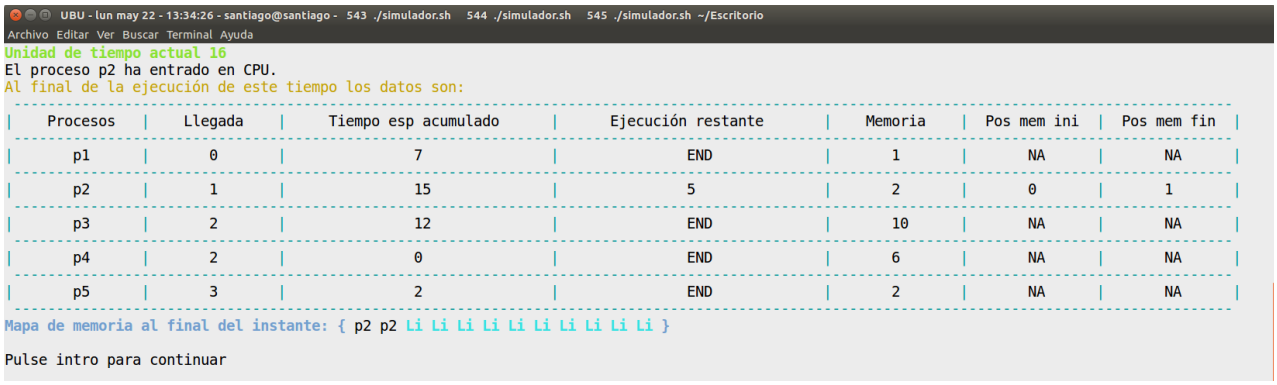
Instante 14, como se decía al acabar P1 se liberó su memoria y por tanto quedo 10 de memoria libre, el gestor de memoria detecta que ahora P3 puede ser alojado, pero surge el problema de que la memoria no está de forma continua, por lo que se reubica, desplazando a P2 a la izquierda. Ahora P3 esta e memoria y por tanto se puede ejecutar ya que es mas corto que P2.



Instante 15, P3 acaba y libera su memoria.



Instante 16, P2 por fin entra en CPU.



Instante 21, P2 acaba sin problemas ya que es el único proceso en lista.

```
UBU - lun may 22 - 13:34:26 - santiago@santiago -  543 ./simulador.sh  544 ./simulador.sh  545 ./simulador.sh ~/Escritorio
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Unidad de tiempo actual 21
Sigue el proceso p2 en CPU.
El proceso p2 retorna al final del tiempo 21, la memoria asignada fue liberada
Al final de la ejecución de este tiempo los datos son:
 ---------------------------------------------------------------------------------------------------------------------
|    Procesos    |    Llegada    |     Tiempo esp acumulado     |      Ejecución restante      |    Memoria    | Pos mem ini | Pos mem fin |
 ---------------------------------------------------------------------------------------------------------------------
|       p1       |       0       |              7               |              END             |       1       |      NA     |      NA     |
 ---------------------------------------------------------------------------------------------------------------------
|       p2       |       1       |              15              |              END             |       2       |      NA     |      NA     |
 ---------------------------------------------------------------------------------------------------------------------
|       p3       |       2       |              12              |              END             |       10      |      NA     |      NA     |
 ---------------------------------------------------------------------------------------------------------------------
|       p4       |       2       |              0               |              END             |       6       |      NA     |      NA     |
 ---------------------------------------------------------------------------------------------------------------------
|       p5       |       3       |              2               |              END             |       2       |      NA     |      NA     |
 ---------------------------------------------------------------------------------------------------------------------
Mapa de memoria al final del instante: { Li Li Li Li Li Li Li Li Li Li Li Li }
Pulsa cualquier tecla para ver resumen...
```

Resumen de tiempos acumulados y de retorno vemos lo el tiempo que ha tenido que esperar cada procesos, así como el tiempo que ha tardado en retornar.

```
UBU - lun may 22 - 14:13:57 - santiago@santiago -  544 ./simulador.sh  545 ./simulador.sh  546 ./simulador.sh ~/Escritorio
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
En la sigueinte tabla se ve para cada proceso el tiempo que ha pasado esperando en la cola de listos.
además del tiempo que ha tardado en retornar una respuesta.
 ---------------------------------------------------
|    Proceso     |        Tiempo Espera Acu     | Tiempo retorno |
 ---------------------------------------------------
|       p1       |              7               |       14       |
 ---------------------------------------------------
|       p2       |              15              |       21       |
 ---------------------------------------------------
|       p3       |              12              |       14       |
 ---------------------------------------------------
|       p4       |              0               |       3        |
 ---------------------------------------------------
|       p5       |              2               |       6        |
 ---------------------------------------------------

santiago@santiago:~/Escritorio$
```

informe220517-1358.txt ← Se genera el informe.

Conclusiónes

Relativas a los conceptos:

SRPT es un planificador que beneficia a los procesos cortos(siempre que haya memoria para alojarlos) y perjudica mucho a los procesos más largos llegando incluso a pasar de ser los primeros en entrar a ser los últimos en salir o directamente en un sistema real donde los procesos son muchísimos más no acabar nunca(morir de inanición).

La gestión de memoria por su parte, el hecho de que tenga que ser continua supone un problema, ya que obliga a tener un servicio de reubicación en segundo plano, otra de las de las conclusiones en este apartado es que este simulador no ha simulado *swapping* por lo que el problema de memoria ocupada se maximiza.

Relativas al simulador:

El hecho de realizar el simulador en un lenguaje como es Shell supone una dificultad a la hora de hacer un código extenso fácil de entender y mantener, además al no ser fácilmente depurable se hace bastante pesado. Otra de las dificultades añadidas es la ausencia de un IDE adecuado para dicho lenguaje. Aunque como conclusión personal es bastante útil conocer este lenguaje, sobre todo si vamos a trabajar en entornos *Unix* y *Unix-Like.*

El simulador es bastante susceptible de mejora tanto en código como en funcionalidad.

Tiempo

Aunque se ha contado con bastante tiempo, la práctica tiene cierta dificultad descrita antes que hace que al final la mayor parte del mismo se consuma en aprender una sisntaxis avanzada de shell lo que al final penaliza bastante el tiempo.

Bibliografía

José Manuel Sáiz. Apuntes asignatura Sistemas Operativos , Grado en Ingeniería Informática 2016-2017,

[**fecha de consulta:** curso 2016-2017]

[**Disponible en:** Campus virtual de la Universidad de Burgos: https://ubuvirtual.ubu.es]

Colaboradores de Wikipedia. Script (informática) [en línea]. Wikipedia, La enciclopedia libre, 2011

[**fecha de consulta:** 11 de mayo del 2017]

[**Disponible en:** https://es.wikipedia.org/w/index.php?title=Script&oldid=98454034]

Colaboradores de Unix & Linux Stack Exchange. Consulta dudas scritpting bash [en línea], Unix & Linux Stack Exchange, web de preguntas y respuestas 2017.

[**fecha de consulta:** mayo del 2017]

[**Disponible en:** https://unix.stackexchange.com/]

Alumnos del Grado en ingeniería informática de la Universidad de Burgos. Código del simulador, *Práctica De control - R.R - Memoria según necesidades - continua y reubicable*

Alumnos:

- José Luis Garrido Labrador
- Mario Juez Gil
- Omar Santos Bernabé
- Luis Pedrosa Ruiz

[**fecha de consulta:** mayo del 2017]

[**Disponible en:** Campus virtual de la Universidad de Burgos: https://ubuvirtual.ubu.es]