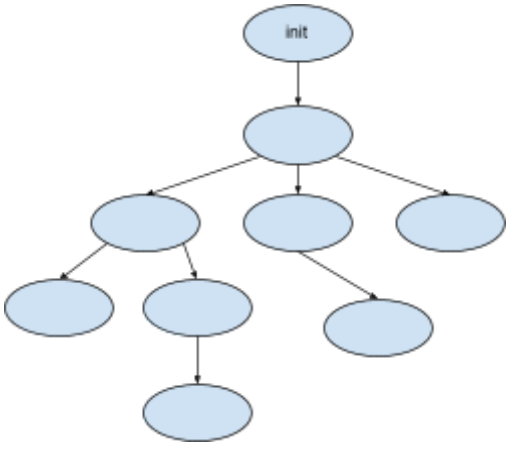
Carlos Segundo Nieto
Héctor Ullate Catalán

Ejercicio 1: Estudia el código del fichero *fork_example.c* y responde a las siguientes preguntas:

- ¿Cuántos procesos se crean? Dibuja el árbol de procesos generado
- ¿Cuántos procesos hay como máximo simultáneamente activos?
- Durante la ejecución del código, ¿es posible que algún proceso quede en estado *zombi*? Intenta visualizar esa situación usando la herramienta *top* e introduciendo llamadas a *sleep()* en el código donde consideres oportuno.
- ¿Cómo cambia el comportamiento si la variable *p_heap* no se emplaza en el *heap* mediante una llamada a *malloc()* sino que se declara como una variable global de tipo *int*?
- ¿Cómo cambia el comportamiento si la llamada a *open* la realiza cada proceso en lugar de una sola vez el proceso original?
- En el código original,¿es posible que alguno de los procesos creados acabe siendo hijo del proceso *init* (PID=1)? Intenta visualizar esa situación mediante *top*, modificando el código proporcionado si es preciso.

- Se crean 8 procesos. El árbol de procesos es el siguiente:



- Como máximo habrá 8 procesos simultáneamente activos.
- Es imposible, puesto que ninguno hace uso de la función exit().
- No se produce ningún cambio en el comportamiento.
- Se escribe únicamente texto del padre y 2 de sus hijos en la iteración 2.
- En el código original no es posible esto. Para conseguirlo, se debe escribir la función sleep en la rama del if para el hijo y eliminar el bucle que espera a la terminación de procesos hijos.

Ejemplo 1. Suma: El código `partial_sum1.c` lanza dos hilos encargados de colaborar en el cálculo del siguiente sumatorio:

$$suma\_total = \sum_{n=1}^{10000} n$$

Después de ejecutarlo varias veces observamos que no siempre ofrece el resultado correcto, ¿Por qué? En caso de no ser así, utiliza el ejemplo codificado en `partial_sum2.c` y observa que nunca obtenemos el resultado correcto. ¿Por qué?

El código *partial_sum1.c* funciona correctamente, puesto que siempre muestra el resultado correcto. En cambio, el código de *partial_sum2.c* no funciona; esto es debido a que en cada iteración del bucle for (de la función *partial_sum*) uno de los 2 hilos va a sobreescribir la suma que ha realizado el otro justo antes, y por lo tanto en cada iteración sólo se sumará al total el valor de uno de los 2 hilos.

Ejercicio 2: Añadir un semáforo sin nombre al **Ejemplo 1 (`partial_sum1.c`)** de forma que siempre dé el resultado correcto. Incluir además la opción de especificar, mediante línea de comando, el valor final del sumatorio y el número de hilos que deberán repartirse la tarea. Ejemplo:

```
./partial_sum1 5 50000
```

sumará los números entre 1 y 50000 empleando para ello 5 hilos.

Se adjunta la solución de este ejercicio en el archivo *ejercicio2.c*

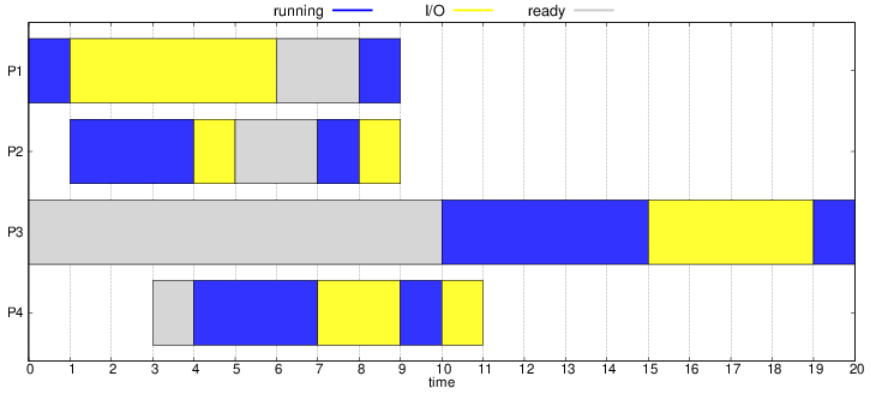
Carlos Segundo Nieto
Héctor Ullate Catalán

Ejercicio 3: escribe un fichero de entrada que simule el conjunto de tareas del ejercicio 7 de la hoja de problemas. Ejecuta el simulador para resolver los apartados *b,c* y *d* de dicho problema.

El fichero txt quedaría así:

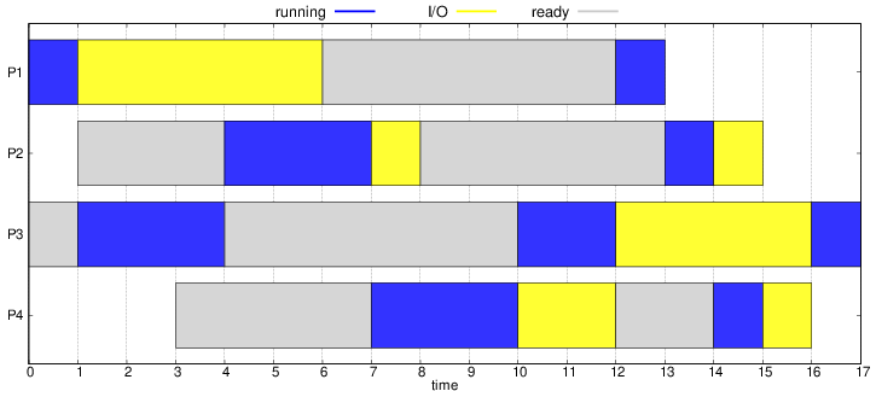
```
P1 1 0 1 5 1
P2 1 1 3 1 1 1
P3 1 0 5 4 1
P4 1 3 3 2 1 1
```

Apartado b (SJF)



Tiempos de ejecución.
P1: 9 ciclos; P2: 8 ciclos; P3: 20 ciclos; P4: 8 ciclos.
Tiempos de espera.
P1: 2 ciclos; P2: 2 ciclos; P3: 10 ciclos; P4: 1 ciclo.
Tiempo promedio de ejecución: 11,25 ciclos.
Tiempo promedio de espera: 7,5 ciclos.
Productividad: 0,2
Porcentaje de uso de CPU del sistema: 100%

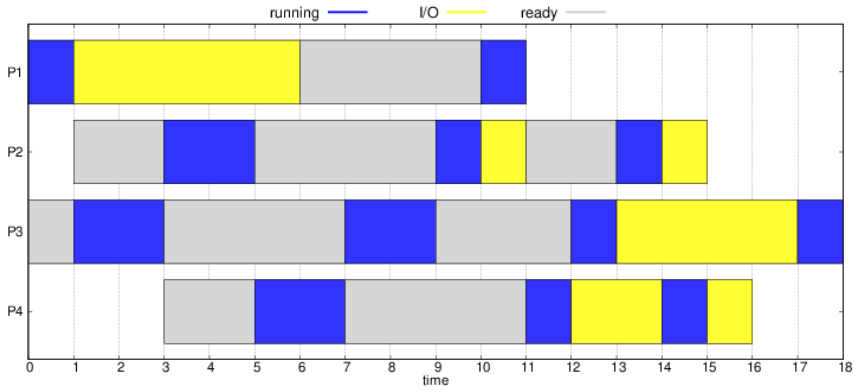
Apartado c (RR con quanto = 3)



Tiempos de ejecución.
P1: 13 ciclos; P2: 14 ciclos; P3: 17 ciclos; P4: 13 ciclos.
Tiempos de espera.
P1: 6 ciclos; P2: 9 ciclos; P3: 7 ciclos; P4: 6 ciclos.
Tiempo promedio de ejecución: 14,25 ciclos.
Tiempo promedio de espera: 7 ciclos.

Productividad: 0,235
Porcentaje de uso de CPU del sistema: 100%

Apartado d (RR con quanto = 2)



Tiempos de ejecución.
P1: 11 ciclos; P2: 14 ciclos; P3: 18 ciclos; P4: 13 ciclos.
Tiempos de espera.
P1: 4 ciclos; P2: 8 ciclos; P3: 8 ciclos; P4: 6 ciclos.
Tiempo promedio de ejecución: 14 ciclos.
Tiempo promedio de espera: 6,5 ciclos.
Productividad: 0,222
Porcentaje de uso de CPU del sistema: 100%

Ejercicio 4: La solución adoptada para evitar interbloqueos en el equilibrador de carga se puede utilizar para resolver el problema de *los filósofos*. Implementa un código que modele dicho problema creando 5 hilos que representarán a los 5 filósofos. Cada uno de ellos se limitará a *pensar* (que se simulará mediante a una llamada a *sleep()* con un tiempo aleatorio) y posteriormente *comer*. Antes de comer, deberán coger dos tenedores: el de su derecha y el de su izquierda (no necesariamente en ese orden...). Posteriormente, dejará los tenedores, dormirá y volverá a pensar. El acto de coger un tenedor se modelará con el intento de adquirir un cerrojo.

Se adjunta la solución de este ejercicio en el archivo *ejercicio4.c*