SISTEMAS OPERATIVOS - LABORATORIO
17 de Enero de 2023

Nombre _____ DNI _____

Apellidos _____ Grupo _____

ADVERTENCIA: Si el código no compila o su ejecución produce un error grave la puntuación de ese apartado será 0.

Cuestión 1. Crear un programa que reciba como argumento el nombre de un fichero de texto (ascii) que describe un grafo dirigido de tareas. Cada línea del fichero tendrá el siguiente formato (todos los campos son números enteros, se separan por coma):

Nombre,Num_predecesores,Predecesor_1,Predecesor_2,...,Predecesor_m

El programa debe parsear este fichero y con la información del fichero rellenar una variable global **tasks** que representa el grafo (**struct task tasks[MAXPROC]**). El array tendrá un tamaño máximo conocido (**#define MAXPROC 16**). Cada elemento del array representa uno de los vértices del grafo (una tarea), usando la siguiente estructura:

```
struct task {
        int valid;              // booleano, indica si la entrada es válida (1) o no (0)
        int next[MAXPROC]; // array de booleanos para sucesores: next[i]=1 si sale una arista hacia el nodo i
        int id;                 // identificador/nombre de la tarea, que corresponde a su posición en el array de nodos
};
```

Para el parseado del fichero puedes considerar el uso de alguna(s) de estas funciones: **fscanf**, **fgets**, **strsep**, **strtol** o **atoi**. Para comprobar el funcionamiento correcto del programa se invocará la función print_graph tras el parsing, que mostrará por pantalla el grafo:

```
void print_graph(struct task tasks[], int n)
{
   int i,j;

   for (i = 0; i < n; i++) {
        if (!tasks[i].valid)
                continue;
        printf("%d: ", tasks[i].id);
        for (j = 0; j < n; j++)
                if (tasks[i].next[j])
                        printf("%d ", j);
        printf("\n");
   }
}
```

Ejemplo: grafo.txt (entrada)	Salida estándar esperada:
0,0 1,1,0 2,1,0 3,1,0 4,2,1,5 5,2,2,3	0: 1 2 3 1: 4 2: 5 3: 5 4: 5: 4

Cuestión 2. Extender el programa anterior para que, tras parsear el grafo e invocar print_graph, cree un hilo POSIX por cada tarea del grafo (nodo del grafo) y espere a que todos los hilos terminen. En la creación cada hilo recibirá como argumento la dirección de un entero que contiene el nombre de la tarea (su posición en el array), puede pasarse la dirección del correspondiente campo id de la estructura struct task. La función de entrada de cada hilo será:

```
void *task_body(void * arg) {
   Int id = *(int *) arg;
   wait_for_predecesors(id);          // bloqueante, el hilo espera a que sus predecesores le notifiquen su finalización
   printf("Task %d running\n", id);  // cuerpo real de la tarea
   notify_successors(id);             // sincronización, aviso a los sucesores de que esta tarea ha terminado
   return NULL;
}
```

Implementar las funciones **wait_for_predecesors()** y **notify_successors()** para que las tareas se sincronicen correctamente. La primera hará que el hilo se quede esperando la notificación de todos sus predecesores en el grafo. La segunda notificará a las tareas sucesoras (las siguientes en el grafo) de la finalización de este hilo/tarea. La sincronización se puede implementar con mutex y variables de condición o con semáforos, a elección del alumno. Deberán añadirse al programa las variables globales y/o campos a la estructura struct task necesarios según el mecanismo de sincronización escogido. Asimismo, se debe añadir al programa principal el código de inicialización de dichos recursos.

NOTA: Si no has conseguido hacer el primer ejercicio (parseado del grafo) puedes incluir la siguiente declaración en tu código para poder resolver este ejercicio:

```
struct task tasks_static[MAXPROC] = {
//                          0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
    {.valid = 1, .next = {0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, .id = 0}, // 0: 1 2 3
    {.valid = 1, .next = {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, .id = 1}, // 1: 4
    {.valid = 1, .next = {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, .id = 2}, // 2: 5
    {.valid = 1, .next = {0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, .id = 3}, // 3: 5
    {.valid = 1, .next = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, .id = 4}, // 4:
    {.valid = 1, .next = {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, .id = 5}, // 5: 4
    0
};
```

Ejemplo:

grafo.txt (entrada)	Ejemplo de salida estándar esperada:
0,0 1,1,0 2,1,0 3,1,0 4,2,1,5 5,2,2,3	0: 1 2 3 1: 4 2: 5 3: 5 4: 5: 4 Task 0 running Task 1 running Task 2 running Task 3 running Task 5 running Task 4 running Las tareas 1, 2 y 3 pueden alterar su orden entre sí, 0 siempre tiene que aparecer antes, mientras que después aparecerán siempre 5 y 4 en este orden.