Práctica 3

Desarrollo extra

3.1. Descripción

Los alumnos incorporarán al **simulador** el planificador **Multinivel (ML)** con tres niveles de prioridad. A su vez, en cada nivel las tareas se ejecutarán por orden de llegada a dicho nivel. Las tareas tienen unas rodajas de tiempos fijos a 1, 2 y 5 ticks para el primer, segundo y tercer nivel. El comportamiento del planificador ML será el siguiente:

- El planificador ejecutará inicialmente aquellas tareas del nivel más prioritario (nivel 1). Si alguna tarea no se pudiese completar en el tiempo asignado al nivel (ej: 1 tick para el nivel 1) automáticamente la tarea migrará al siguiente nivel de prioridad.
- Cuando una tarea nueva llega al sistema, esta se planificará en el nivel más prioritario no siendo necesario expropiar ningún trabajo en curso aunque este estuviese en una cola menos prioritaria.
- El nivel menos prioritario (nivel 3) se basa en una planificación circular del tipo `round-robin`, si no pudiese ser completada en el tiempo asignado (5 ticks) pasaría al final de la cola de ejecución.

La invocación del nuevo planificador se efectuará de manera análoga a la práctica obteniéndose el esquema de planificación (ver figura 3.1):

```
carlos@posets:~$ ./schedsim -i examples/exampleML.txt -n 1 -s ML
```

3.2. Desarrollo de la parte opcional

A continuación se describirán las tareas a realizar para llevar a cabo con éxito codificación del planficador **ML** para simulador:

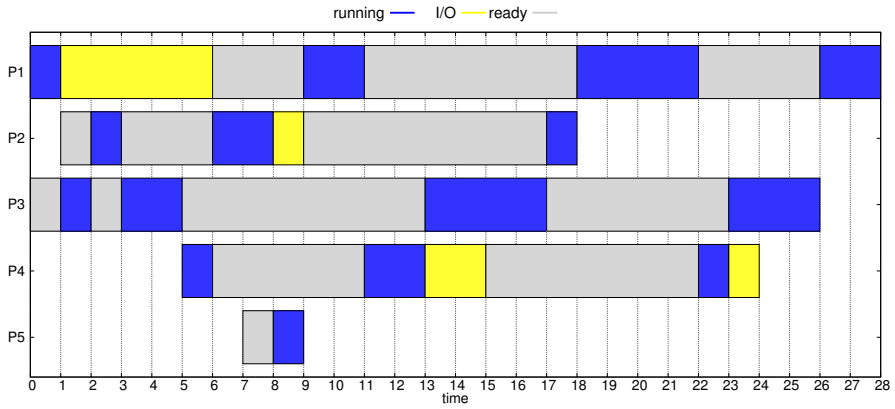

Figura 3.1: Simulación del planicador ML para el ejemplo `exampleML.txt`

1. **[2 puntos]** Dar de alta el planificador ML en el simulador. Se recomienda que todas la modificaciones se efectúen en el fichero **sched.h** y se añada al proyecto el nuevo archivo **sched_ml.c**.

2. **[6 puntos]** Implementación del planificador **ML**. Para ello se recomienda seguir los siguientes pasos:

   - Basándose en el planificador `RR`, incorporar los datos privados referente a las tareas del planificador ML definiendo la estructura `ml_data` que refleje tanto los ticks pendientes como el nivel en el que se encuentra la tarea. Dicha estructura debería ser apuntada (y creada) por el campo `tcs_data` de la estructura `task_t` descrita en el fichero **sched.h**. Basándose en la función `task_new_rr`, se recomienda adecuar la funcionalidad de `task_new_ml` para actualizar la codificación de dicha función.

     ```
     struct ml_data {
        int remaining_ticks_slice_per_level;
        int level;
     };
     ```

   - Basándose en la función `enqueue_task_prio` del planificador de prioridad, la función `enqueue_task_ml` debería de ordenar las tareas de la `runqueue` en función del nivel en el que se encuentren.
   - Basándose en la función `task_tick_rr` del planificador RR, la función `task_tick_ml` debería considerar que cuando agota los ticks disponibles de dicho nivel debería planificar la tarea en el siguiente nivel de prioridad.

3. **[2 puntos]** Implementar la barrera mediante semáforos en el fichero **barrier.c**. Para ello sustituir el contenido de la estructura `sys_barrier_t` del fichero **barrier.h** por las siguientes líneas de código:

   ```
   /* Synchronization barrier */
   typedef struct {
      sem_t mutex; /* Barrier lock */
   ```

```
    sem_t cond_slave; /* Variable where threads remain blocked */
    sem_t cond_master; /* Handshake master-slave */
    int nr_threads_arrived; /* Number of threads that reached the barrier */
    int max_threads; /* Number of threads that rely on the barrier */
} sys_barrier_t;
```