Sistemas Operativos

Práctica 3

Curso

2017-2018

Procesos e hilos: planificación y sincronización

J. C. Sáez

Índice

Introducción

- Objetivos:
  - Comprender la estructura de un planificador a través de un simulador realista
  - Comprender el funcionamiento de las políticas de planificación básicas
  - Implementar diferentes políticas de planificación en un entorno de simulación realista
- Como objetivo instrumental, el alumno se familiarizará con el uso de:
  - POSIX Threads
  - Semáforos
  - Mutexes y variables condición

Introducción

- Recordad: Procesos vs. Hilos
  - Dos procesos (padre - hijo) *no comparten nada*: se duplica toda la imagen de memoria
  - Dos hilos (mismo proceso) *comparten todo* menos la pila
- Haced y estudiad los ejercicios y ejemplos
  - Ayudan a comprender la materia. . . .
  - . . . y suelen acabar en las cuestiones

Índice

Ejemplo de uso I

Terminal #1

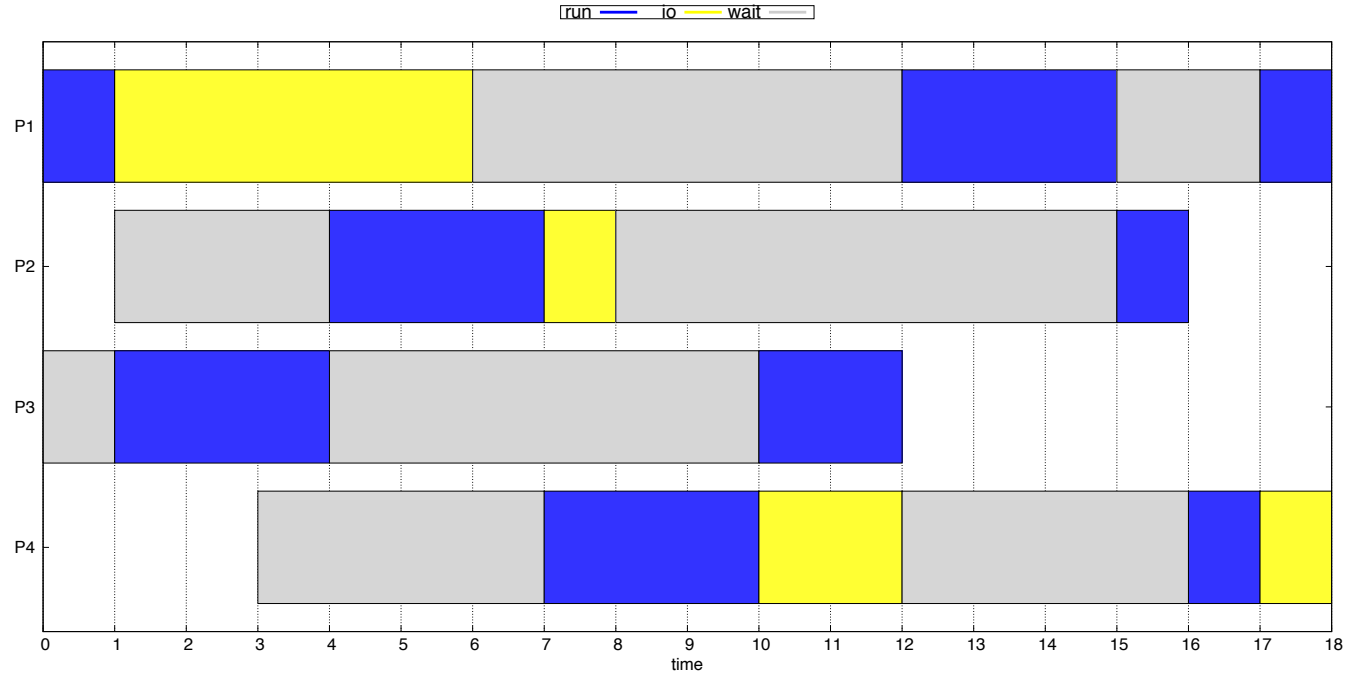
```
debian:P3 usuarioso$ ./schedsim -h
Usage: ./schedsim -i <input-file> [options]

List of options:
  -h: Displays this help message
  -n <cpus>: Sets number of CPUs for the simulator (default 1)
  -m <nsteps>: Sets the maximum number of simulation steps (default 50)
  -s <scheduler>: Selects the scheduler for the simulation (default RR)
  -d: Turns on debug mode (default OFF)
  -p: Selects the preemptive version of SJF or PRIO (only if they are selected with -s)
  -t <msecs>: Selects the tick delay for the simulator (default 250)
  -q <quantum>: Set up the timeslice or quantum for the RR algorithm (default 3)
  -l <period>: Set up the load balancing period (specified in simulation steps, default 5)
  -L: List available scheduling algorithms
```

Ejemplo de uso II

Terminal #1

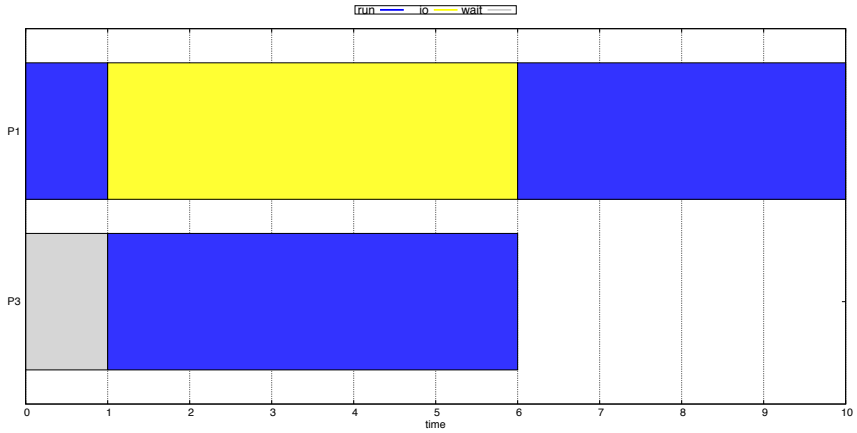
```
debian:P3 usuarioso$ ./schedsim -i examples/example1.txt
debian:P3 usuarioso$ cd ../gantt-gplot
debian:P3 usuarioso$ ./generate_gantt_chart ../schedsim/CPU_0.log
```

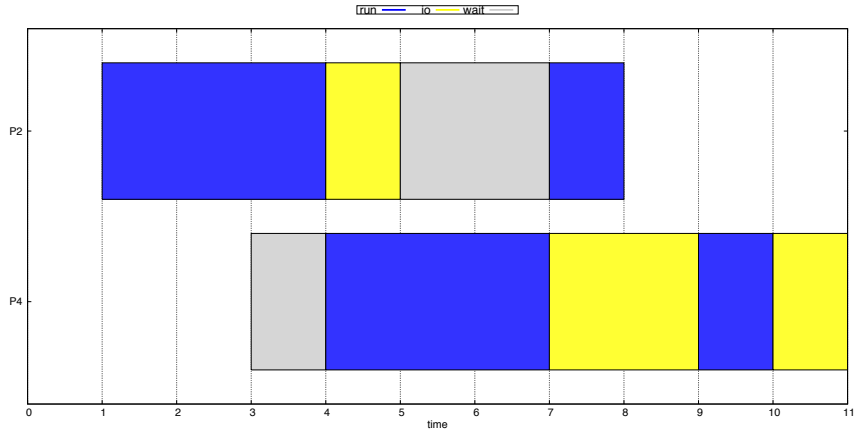


Ejemplo de uso con 2 CPUs

Terminal #1

```
debian:P3 usuarioso$ ./schedsim -i examples/example1.txt -n 2
debian:P3 usuarioso$ cd ../gantt-gplot
debian:P3 usuarioso$ ./generate_gantt_chart ../schedsim/CPU_0.log
debian:P3 usuarioso$ ./generate_gantt_chart ../schedsim/CPU_1.log
```





Sintaxis de ficheros de tareas

- En la carpeta *examples* se incluyen varios ejemplos
- Es sencillo construir nuevos ejemplos siguiendo la sintaxis

```
$ cat examples/example1.txt
P1 1 0 1 5 4
P2 1 1 3 1 1
P3 1 0 5
P4 1 3 3 2 1 1
```

- Columna 1: nombre de la tarea
- Columna 2: prioridad
- Columna 3: tiempo de comienzo
- Columnas siguientes:
  ráfaga CPU - bloqueo - ráfaga CPU ....

Índice

Características generales

- Simulador diseñado para evaluar estrategias de planificación basado en eventos
  - El tiempo está discretizado y simulado por *ticks*
  - En cada *tick*:
    - Eventos: NEW_TASK o TASK_AWAKEN
    - El planificador decide si hay que cambiar la tarea en ejecución
- No se simula lo que hace cada tarea
- Se cuenta del nº de ticks en ejecución y bloqueo
- Simula un multiprocesador, un hilo por CPU
  - Cada CPU tiene su propia *runqueue*
  - Hay un balanceo de carga periódico

Índice

Cola de tareas

runqueue_t

```
typedef struct{

  slist_t tasks;        /* runnable task queue */
  task_t* cur_task;     /* Pointer to the task in the CPU.
                           It may be the idle task*/
  task_t idle_task;     /* This CPU's idle task */
  bool need_resched;    /* Flag activated when a user preemption
                           must take place */
  int cpu_rq;           /* CPU associated with this run queue */
  int nr_runnable;      /* Keeps track of the number of runnable
                           task in this CPU. Current not on the RQ*/
  void* rq_cs_data;     /* Pointer enabling a scheduling class to
                           store private data if needed */
  int next_load_balancing;  /* Time of the next sim. step where
                               load_balancing will take place */
  pthread_mutex_t lock;     /* Runqueue lock*/

}runqueue_t;
```

Tarea I

task_t

```
typedef struct{

  int task_id;                          /* Internal ID for the task*/
  char task_name[MAX_TASK_NAME];
  exec_profile_t task_profile;          /* Task behavior */
  int prio;
  task_state_t state;
  int last_cpu;                     /* CPU where the task ran last time */
  int last_time_enqueued;           /* Last simulation step where the task
                                       was enqueued */
  int runnable_ticks_left;          /* Number of ticks the application has
                                       to complete till block or exit */
  list_node_t ts_links;             /* Node for the global task list */
  list_node_t rq_links;             /* Node for the RQ list */
  ...
```

Tarea II

task_t

```
...
  bool on_rq;                   /* Marker to check if the task is on
                                   the rq or not !! */
  unsigned long flags;          /* generic flags field */
  void* tcs_data;               /* Pointer enabling a scheduling class
                                   to store private data if needed */
  /* Global statistics */
  int user_time;              /* Cpu time */
  int real_time;              /* Elapsed time since the application
                                 entered the system */
  int sys_time;               /* For now this time reflects the time
                                 the thread spends doing IO */
  slist_t sched_regs;         /* Linked list to keep track of the sched
                                 log registers (track state changes for
                                 later use) */
} task_t;
```

Operaciones sobre listas I

Métodos a emplear del tipo `slist_t`

```
void* head_slist ( slist_t* slist );
void* tail_slist ( slist_t* slist );
void remove_slist ( slist_t* slist,void* elem );
void insert_slist ( slist_t* slist,void* elem );

void sorted_insert_slist( slist_t* slist, void* object,
                          int ascending,                    /*TRUE*/
                          int (*compare)(void*,void*) );

void sorted_insert_slist_front( slist_t* slist, void* object,
                          int ascending,                    /*TRUE*/
                          int (*compare)(void*,void*) );
```

Operaciones sobre listas II

Métodos extra del tipo `slist_t`

```
void init_slist   ( slist_t* slist, size_t node_size,
void destroy_slist( slist_t* slist );
void* next_slist  ( slist_t* slist,void* elem);

void insert_slist_head ( slist_t* slist,void* elem);
                         size_t offset_node_field);

void insert_after_slist( slist_t* slist, void *object,
                         void *nobject);

void insert_before_slist( slist_t* slist, void *object,
                          void *nobject);

void sort_slist( slist_t* slist, int ascending,
                 int (*compare)(void*,void*) );

static inline int is_empty_slist(slist_t* slist)
  { return slist->size==0; }

static inline int size_slist(slist_t* slist)
  { return slist->size; }
```

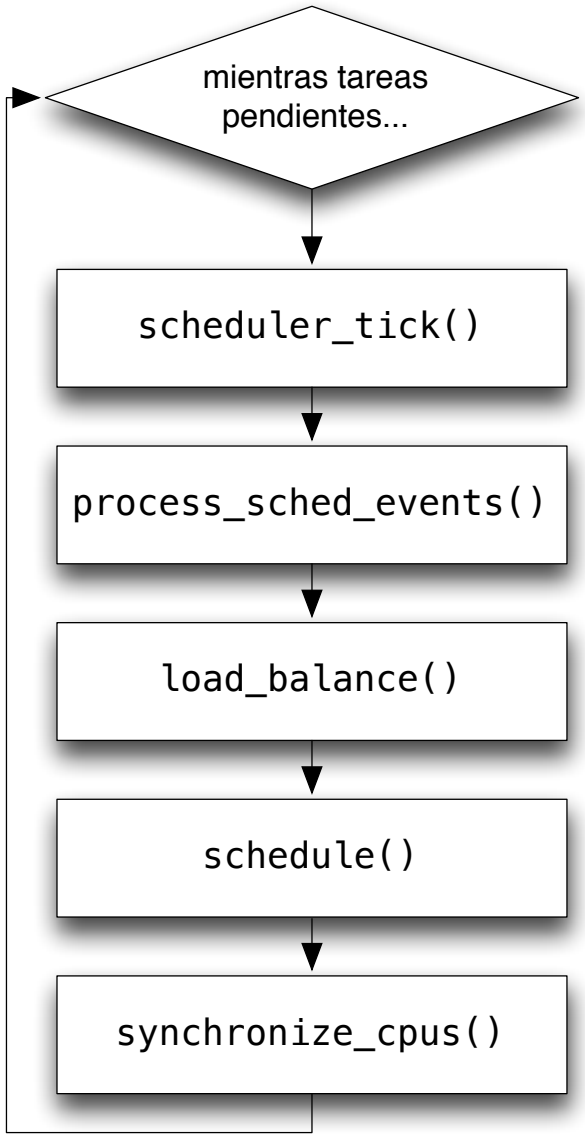
Índice

Inicio del Simulador

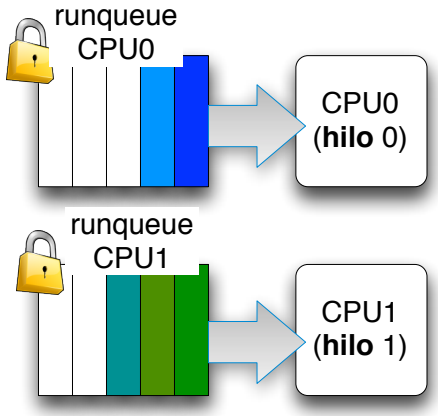
Se carga la lista de tareas y se invoca `sched_start()`

1. Inicializa las estructuras globales `sched_init()`
   - Array de `runqueues`: una por CPU
   - Array de eventos `sched_events`: una por CPU
   - Llama al *callback* de inicialización de la política elegida
   - Otras estructuras (mutex, ...) e inicializaciones
2. Recorre la lista de tareas, y para cada una
   - Inicializa la tarea
   - Inserta las tareas en las *runqueue* de las CPUs (reparto circular + NEW_TASK)
3. Crea un hilo por cada CPU (función `sched_cpu()`)
4. Espera a que terminen los hilos
5. Escribe a disco los resultados (log)
6. Libera los recursos y termina

Ciclo de simulación



- Cada hilo asociado a cada CPU realiza el bucle mientras le queden tareas pendientes
- Una iteración del bucle es equivalente a un *tick* del planificador
  - Cada CPU (hilo) tiene su cola de tareas listas para ejecutar (*runqueue*)
  - Cada *runqueue* tiene un cerrojo para evitar accesos concurrentes desde otras CPUs



Accounting de tarea actual

`scheduler_tick()`

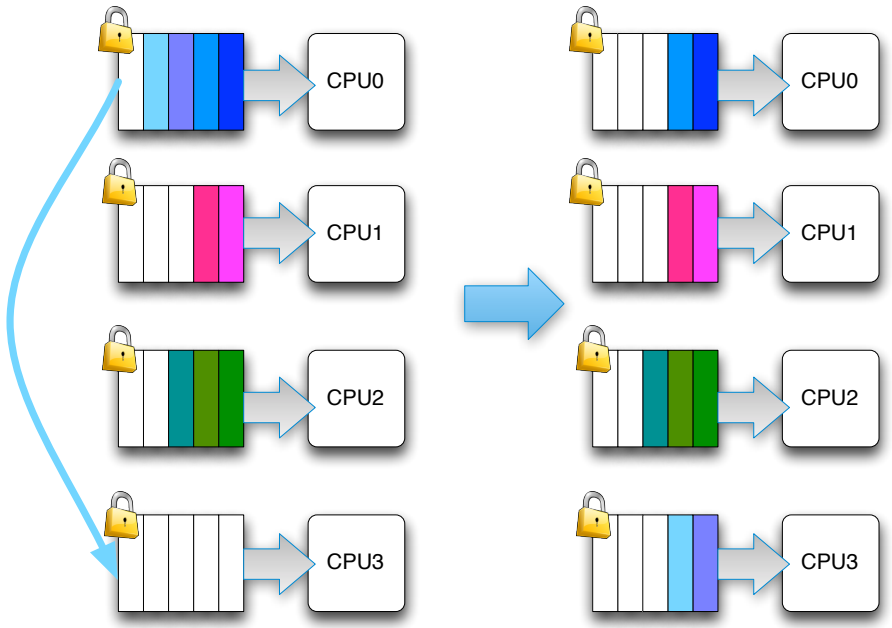
1. Actualiza estadísticas:
   - Incrementa tiempo de ususario y real
   - Marca la última CPU en la que se ejecutó
2. Invoca, si existe, el método `task_tick()` de la política activa
3. Si la tarea actual no es IDLE y ha terminado una fase:
   - Marca la cola para replanificación
   - Selecciona la nueva fase para la tarea actual
     - Debe ser una fase de E/S
     - Actualiza sus estadísticas (tiempos de sistema y real incrementados en la longitud de la fase de E/S)
     - Inserta evento TASK_AWAKEN para el final de la nueva fase

Procesamiento de eventos

process_sched_events()

1. Evento NEW_TASK: aparece una nueva tarea
   - La tarea se inicializa y se añade a la cola (*runqueue*) usando el método enqueue_task de la política activa
2. Evento TASK_AWAKEN: una tarea termina una fase de E/S
   - Si ya no quedan fases por ejecutar se *termina* la tarea
   - Si no, se actualiza la fase de la tarea (debe ser de CPU)
     - ticks que dura la fase
     - tarea marcada como lista
     - ...
   - Se inserta en la cola (enqueue_task)

Balanceo de carga



- Se ejecuta periódicamente o si una *runqueue* está vacía
- El hilo de la CPU más cargada trata de llevar trabajo a la CPU más descargada (`enqueue_task`)
- El hilo de la CPU menos cargada trata de *robar* tareas a la CPU más cargada (`steal_task`)
- Puede ocurrir en paralelo: posible *deadlock*
  - Implementación específica, similar al *Problema de los filósofos*

Plafificación

`schedule()`

- Se replanificará si:
  - el flag `need_resched` está activo
  - la tarea actual es `IDLE`
- El flag `need_resched` estará activo si:
  - La política de planificación ha decidido expropiar la CPU
    - Ha llegado a la *runqueue* una tarea con más prioridad
    - La tarea actual ha agotado su cuanto
    - ...
  - La tarea actual ha pasado a fase de E/S o ha terminado (lo marca el simulador)
- En cualquier caso la tarea se escoge invocando el método `pick_next_task` de la política activa

Índice

Implementando planificador

Métodos que se deben implementar

```
void sched_init_<name>( void );
void sched_destroy_<name>( void ) ;
static void task_new_<name>( task_t* t );
static void task_free_<name>( task_t* t );
static task_t* pick_next_task_<name>( runqueue_t* rq ) ;
static void enqueue_task_<name>( task_t* t, runqueue_t* rq,
                                 int preempted );
static void task_tick_<name>( runqueue_t* rq );
static task_t* steal_task_<name>( runqueue_t* rq );
```

`sched_init` y `sched_destroy`

- Funciones de inicialización y destrucción del planificador
- Invocadas sólo una vez en el programa, en la fase inicialización
- Opcionales. Sólo son necesarias si el planificador necesita almacenar información extra en el campo `rq_cs_data` de la runqueue
  - en tal caso `sched_init()` es responsable de reservar la memoria
  - y `sched_destroy()` de liberarla

`task_new` y `task_free`

- Parámetro de entrada:
  - `task_t* t`
- Invocadas al crear y destruir una tarea
- Opcionales. Sólo son necesarias si el planificador tiene que mantener datos extra para las tareas en el campo `tcs_data`
  - en tal caso `task_new()` será responsable de reservar la memoria
  - y `task_free()` de liberarla
- Ejemplo: Round Robin debe almacenar el cuanto para la tarea en el campo `tcs_data`

pick_next_task

- Obligatoria
- Parámetros de entrada:
  - `runqueue_t* rq`
- Se invoca cuando se deba escoger la siguiente tarea a ejecutar en la CPU
- Sus funciones serían:
  - Escoger una tarea de la `runqueue`
  - Sacar dicha tarea de la `runqueue`
  - Devolver dicha tarea

enqueue_task

- Obligatoria
- Parámetros de entrada:
  - `task_t* t`
  - `runqueue_t* rq`
  - `int preempted`
- Se invoca para encolar una tarea en la `runqueue` de una CPU
- Con `preempted = 1`, tarea expulsada:
  - Se estaba ejecutando y hay que replanificar (e.g. agota cuanto).
- Con `preempted = 0`, tarea nueva. Si la política es expropiativa, debe comprobar si es necesario replanificar y marcarlo. Situaciones:
  - Porque acaba de crearse
  - Porque se ha desbloqueado (terminada su fase de E/S)
  - Porque se ha migrado desde otra CPU

task_tick

- Parámetros de entrada:
  - `runqueue_t* rq`
- Invocada en el bucle principal, una vez por *tick* de simulación
- Opcional. Realiza el accounting propio de la pólitica si es necesario
  - Ejemplo: en RR decrementa el número de *ticks* que faltan para completar el cuanto
- Marca la cola para replanificación en caso de que sea necesario
  - Ejemplo: en RR si se ha agotado el cuanto

steal_task

- Obligatoria
- Parámetros de entrada:
  - `runqueue_t* rq`
- Se invoca cuando sea necesario hacer una migración que implique *robar* una tarea de la `runqueue`.

Campos que debe modificar la política

- De `runqueue_t`
  - `tasks`: la lista de tareas
  - `need_resched`: flag para indicar la necesidad de replanificar
  - `cs_data`: información extra del planificador
- De `task_t`
  - `flags`: para añadir flags interesantes para la política (ver por ejemplo la implementación de SJF)

Registro de la clase de planificación

- Añadir un nuevo identificador numérico a la política en `sched.h`

```
/* Numerical IDs for the available scheduling algorithms */
enum {
  RR_SCHED,
  SJF_SCHED,
  NR_AVAILABLE_SCHEDULERS
};
```

- Añadir también nueva entrada en la estructura `available_schedulers`
  - Identificador numérico
  - Nombre asignado a la política
  - Dirección de la variable `sched_class_t` que la implementa

```
static const sched_choice_t available_schedulers[
  NR_AVAILABLE_SCHEDULERS]={
    {RR_SCHED,"RR",&rr_sched},
    {SJF_SCHED,"SJF",&sjf_sched},
};
```

Índice

Round Robin y SJF

- Ver código fuente

Índice

Parte obligatoria

Cambios en el código C

- Crear planificador FCFS **no expropiativo**
- Crear planificador **expropiativo** basado en prioridades
- Implementar una barrera de sincronización no reentrante usando cerrojos y VC
  - Completar el fichero `barrier.c` (funciones `sys_barrier_init()`, `sys_barrier_destroy()` y `sys_barrier_wait()` de la rama `#else`)
  - Cuidado con los despertares espurios de la llamada `pthread_cond_wait` (usar `while`)
  - Modificar el *Makefile* para evitar que se declare la macro `POSIX_BARRIER`

Script shell

- No recibirá argumentos de entrada pero solicitará al usuario:
  - Nombre del fichero de ejemplo que se desea simular
  - Número máximo de CPUs que se desean simular
- Se simulará el ejemplo para todos los planificadores y todos los números de CPUs posibles (hasta el máximo indicado)
  - Para cada uno, se generarán las gráficas correspondientes

Pseudo código script shell

Script shell

```
maxCPUs = valor introducido por usuario
foreach nameSched in  listaDeSchedulersDisponibles do
    for cpus = 1 to maxCPUs do
        ./simulador -n cpus -i .......
        for i=1 to cpus  do
            mover CPU_$i.log a  resultados/nameSched-CPU-$i.log
        done
        generar gráfica
    done
done
```