Sistemas Operativos

Curso

2014-2015

Práctica 3

Procesos e hilos: planificación y sincronización

Profesores Sistemas operativos

Introducción

Objetivo

- El objetivo principal de la práctica es 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, variables condición

Introducción

Recordad: Procesos vs. Hilos

- Dos procesos (padre - hijo) *no comparten nada* : se duplica toda la imagen de memoria
- Dos hilos (del mismo proceso) *comparten todo* menos la pila

Mecanismos de sincronización

- Cerrojos
- Variables condicionales
- Semáforos

Haced los ejercicios / ejemplos

- Ayudan a comprender la materia....
- ... y suelen acabar en las cuestiones

Ejemplo de uso (1)

Terminal #1

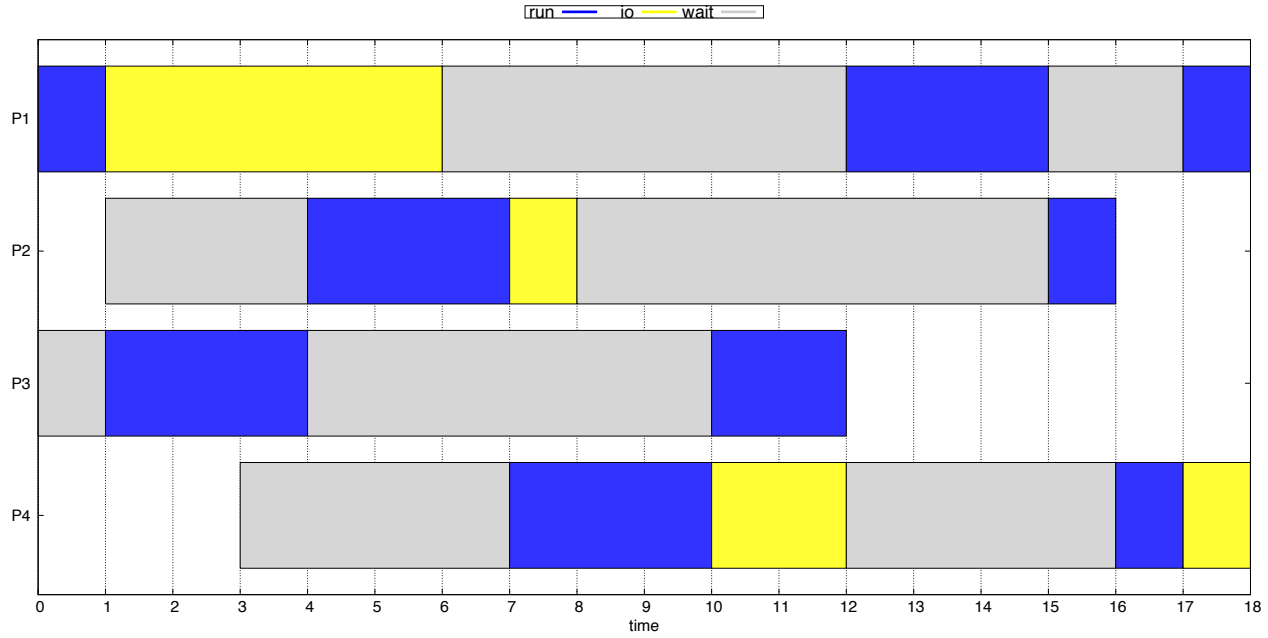
```
debian:P3 usuarioso$ ./schedsim -h
Usage: ./schedsim -i <input-file> [options]

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 (2)

Terminal #1

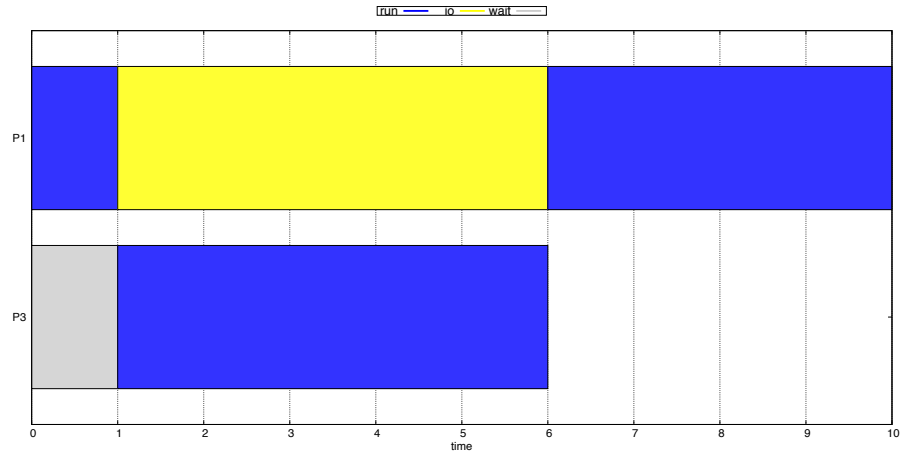
```
debian:P3 usuarioso$ ./schedsim -i examples/example1.txt
debian:P3 usuarioso$ cd ../gantt-gplot
debian:P3 usuarioso$ ./generate_gantt_chart ../schedsim/CPU_0.log
```

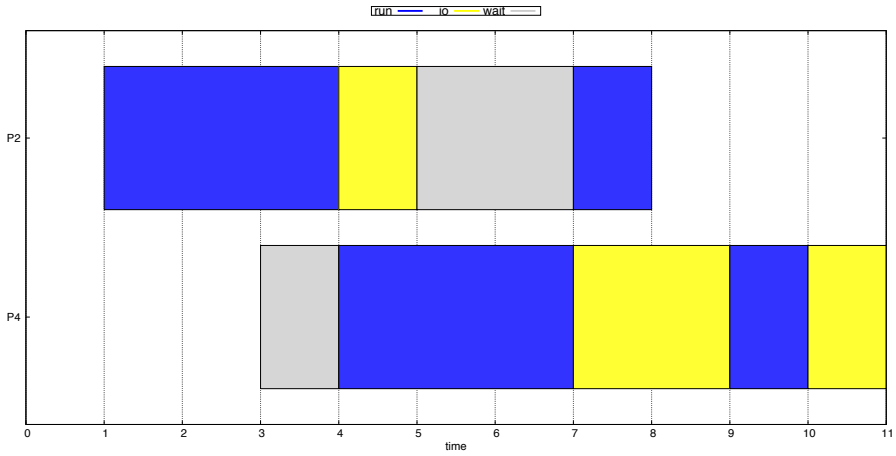


Ejemplo de uso con 2 CPUs

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

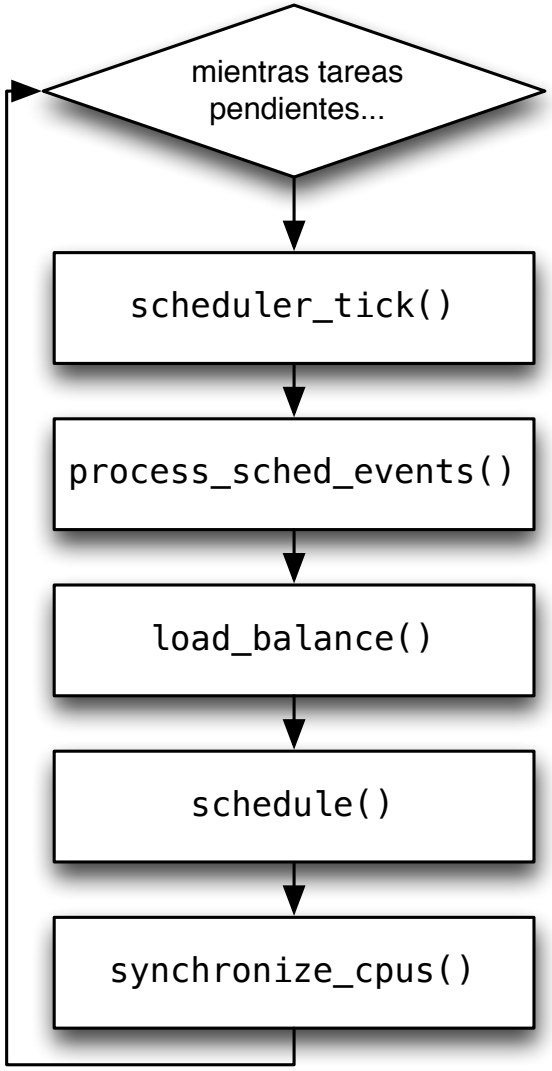
Ejemplos proporcionados

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

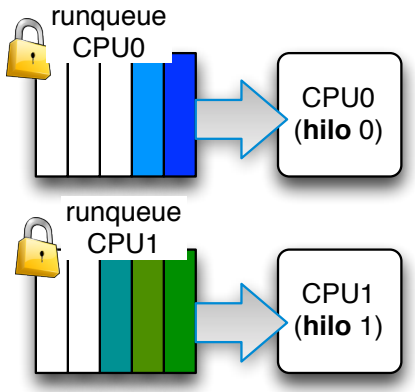
```
$ cat examples/example1.txt
P1 1 0 1 5 4
P2 1 1 3 1 1
P3 1 0 5
P4 1 3 3 2 1 1
```

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

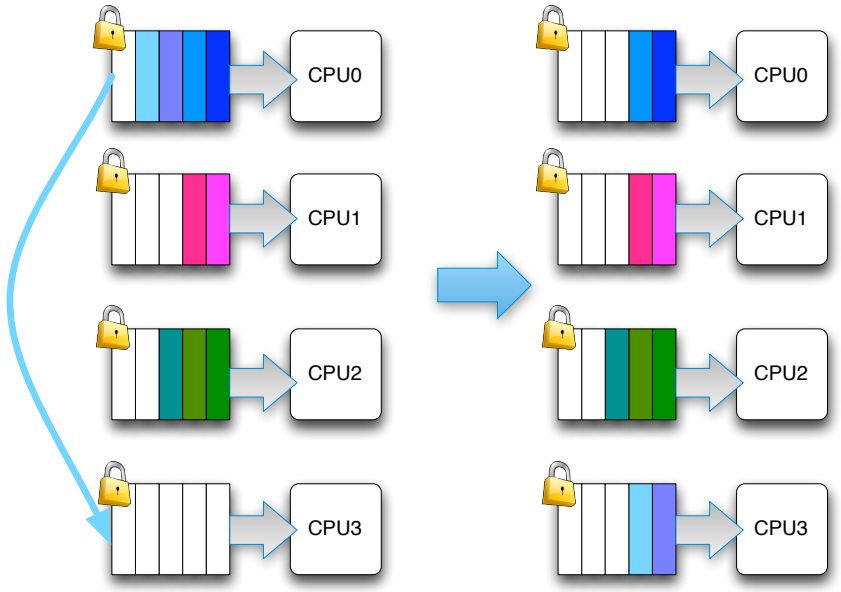
Ciclo de simulación



- Existe un hilo real por cada CPU simulada
- Cada hilo realiza este 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



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
- El hilo de la CPU menos cargada trata de *robar* tareas a la CPU más cargada
  - Puede ocurrir en paralelo: posible *deadlock*
  - Implementación específica, similar al *Problema de los filósofos*

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 nr_runnable;    /* Keeps track of the number of runnable task in this CPU
                           -> Note that current is not on the RQ */
    int next_load_balancing; /* Timestamp of the next simulation step
                                where load_balancing will take place */
    pthread_mutex_t lock;    /* Runqueue lock*/
}runqueue_t;
```

Tareas

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 blocking or exiting */
  int remaining_ticks_slice; /* For the RR scheduler */
  ...
  bool on_rq;               /* Marker to check if the task is on the rq or not !! */
  unsigned long flags;      /* generic flags field */

  /* Global statistics */
  int user_time;      /* Cpu time */
  int real_time;      /* Elapsed time since the application entered the system */
  int sys_time;       /* Reflects the time the thread spends doing IO */
  ...
  }task_t;
```

Operaciones sobre listas

Métodos del tipo `slist_t`

```
extern void insert_slist( slist_t* slist, void* elem );
extern void insert_slist_head( slist_t* slist, void* elem );
extern void remove_slist( slist_t* slist, void* elem );
extern void* head_slist( slist_t* slist );
extern void* tail_slist( slist_t* slist );
void sorted_insert_slist( slist_t* slist, void* object, int ascending,
                          int (*compare)(void*,void*) );
void sorted_insert_slist_front( slist_t* slist, void* object, int ascending,
                                int (*compare)(void*,void*) );
```

Implementando un 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 task_t* pick_next_task_<name>(runqueue_t* rq,int cpu) ;
static void enqueue_task_<name>(task_t* t,int cpu, int runnable);
static void task_tick_<name>(runqueue_t* rq,int cpu);
static task_t* steal_task_<name>(runqueue_t* rq,int cpu);
```

Registrando el planificador RR

Variable sched_class

```
sched_class_t rr_sched={
    .sched_init      = sched_init_rr,
    .sched_destroy   = sched_destroy_rr,
    .task_new        = task_new_rr,
    .pick_next_task = pick_next_task_rr,
    .enqueue_task    = enqueue_task_rr,
    .task_tick       = task_tick_rr,
    .steal_task      = steal_task_rr
};
```

Añadir entrada en available_schedulers

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

Parte obligatoria

Cambios en el código C

- Crear planificador FCFS **no expropiativo**
- Crear planificador **expropiativo** basado en prioridades
- Implementar una barrera de sincronización usando cerrojos y variables condicionales
  - Completar el fichero `barrier.c` (funciones `sys_barrier_init()`, `sys_barrier_destroy()` y `sys_barrier_wait()` de la rama `#else`)
  - 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
```