Simusched - Trabajo Práctico N°1

Sistemas Operativos - 2011 "Año del conejo"
¿Año en que ivissani se licencia?

adeymo ⊕ ivissani ⊕ dfslezak

"Y ahora tiro yo porque me toca"

¿Para qué *schedulear* (calendarizar)?

- Para optimizar una o más medidas de *performance* de mi sistema, posiblemente contradictorias.
- Para tener algún criterio para decidir qué proceso ejecuto a continuación.
- Para simular la ejecución simultánea de N tareas con M CPUs ($M << N$).

A esta altura deberían saber...

- Que los *schedulers* pueden ser *preemptive* (con desalojo) o *non preemptive* (cooperativos).
- Que no existe el *scheduler* óptimo. Todo depende del contexto (acostúmbrense).
- A grandes rasgos los siguientes algoritmos: FCFS (con y sin desalojo), SJF, Round Robin, Multilevel Feedback Queue.
- Que ningún SO real implementa un *scheduler* tan sencillo. En general se usan "combinaciones".

¿Y todo esto qué tiene que ver con el TP?

Nosotros les damos un simulador (en C++ ¿o pensaron que iban a zafar?)

Ustedes van a tener que programar algoritmos de *scheduling* y analizarlos

En concreto: `<enunciado>`

tar zxvf /dev/random > simusched

Simusched es un simulador de *scheduling* exclusivo de la cátedra de SO 2011.

- Se compila haciendo
  `$ make`
- Se ejecuta haciendo
  `$ ./simusched <archivo_tareas.tsk> <costo_cs> <sched> [<params_sched>]`

Defina su *scheduler* en 3 simples pasos

1. Escriba una clase en C++ que herede de la clase `SchedBase`
2. Implemente el constructor de la clase y los métodos
   `load(pid)`,`unblock(pid)`,`tick(motivo)`
   de dicha clase.
3. Agregue la línea
   `_sched_create(<ClaseScheduler>, <CantidadParamsConstructor>)`
   en la función `sched_create()`
   del archivo `main.cpp`

¡Ojo!: No se olvide de modificar el `Makefile` según necesite.

Ej: SchedFCFS.h

```
#ifndef __SCHED_FCFS__
#define __SCHED_FCFS__

#include <vector>
#include <queue>
#include "basesched.h"

class SchedFCFS : public SchedBase {
    public:
        SchedFCFS(std::vector<int> argn);
        virtual void load(int pid);
        virtual void unblock(int pid);
        virtual int tick(const enum Motivo m);

    private:
        std::queue<int> q;
};

#endif
```

Ej: SchedFCFS.cpp

```
SchedFCFS::SchedFCFS(vector<int> argn) {
    // FCFS no recibe par'ametros.
}

void SchedFCFS::load(int pid) {
    q.push(pid); // llegó una tarea nueva
}

void SchedFCFS::unblock(int pid) {
    // Uy! unblock!... bueno, ya seguir'a en el próximo tick
}

int SchedFCFS::tick(const enum Motivo m) {
    if (m == EXIT) {
        // Si el pid actual terminó, sigue el próximo.
        if (q.empty()) return IDLE_TASK;
        else {
            int sig = q.front(); q.pop();
            return sig;
        }
    } else {
        // Siempre sigue el pid actual mientras no termine.
        if (current_pid() == IDLE_TASK && !q.empty()) {
            int sig = q.front(); q.pop();
            return sig;
        } else {
            return current_pid();
        }
    }
}
```

¿Y las tareas? ¿Y la moto? ¿Y Candela?

- Los tipos de tarea son simples funciones `C++`...
- ...definidas en `task.cpp` ...
- ...y registradas en la función `tasks_init()` mediante un llamado a `register_task(<funcion>, <#parametros>);`

Ej: tasks.cpp

```
void TaskCPU(vector<int> params) { // params: n
    uso_CPU(params[0]); // Uso el CPU n milisegundos.
}

void tasks_init(void) {
    /* Todos los tipos de tareas se deben registrar acá para
     * poder ser usadas. El segundo parámetro indica la cantidad
     * de parámetros que recibe la tarea como un vector de enteros,
     * o -1 para una cantidad de parámetros variable. */
    register_task(TaskCPU, 1);
}
```

Entonces podemos hacer cosas tan complejas como queramos. Pero si hacemos un `while (true) {}` colgamos el simulador. Así que no lo hagan.

Es decir que la única condición es que todos los tipos de tarea empiecen y terminen (en algún momento).

Todo muy rico, peeero...

¿Cómo hago para hacer algo útil con todo esto?

Escribiendo archivos `.tsk` que definen conjuntos de tareas.

Por ejemplo:

```
TaskIO 6 2
TaskIO 6 1
@6:
TaskCPU 11
TaskCPU 8
@4:
TaskCPU 7
@60:
*3 TaskCPU 7
```

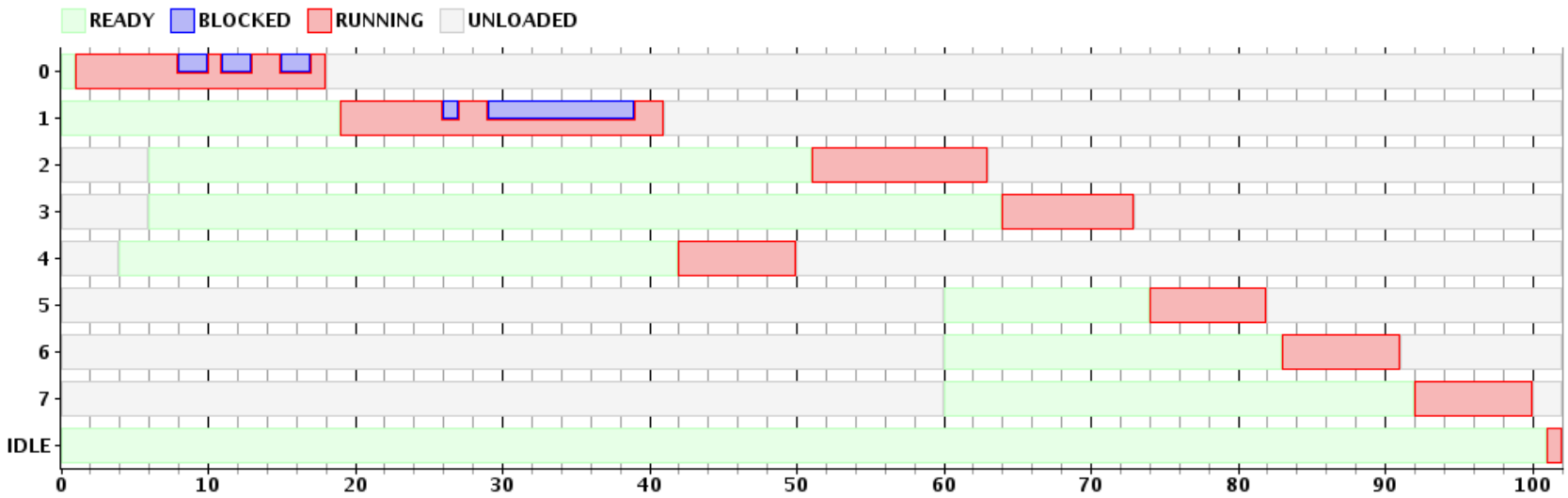
Por si esto fuera poco

En esta inigualable oferta, señora, señó, usted se lleva GRATIS un *script* en `python` que toma la salida de `simusched` y hace el Gantt de la ejecución.

¿Cómo?

```
$ ./simusched <f.tsk> <cs> <sched> [<params>] | \
python graphsched.py > <salida.png>
```

Y eso nos da cosas como:



Deben enviar el código debidamente comentado y el informe, a `sisopdc@gmail.com` con subject:

```
TP 1 Scheduling: grupo Apellido1, Apellido2, Apellido3
```

Reemplazando Apellido$_i$ por los apellidos de los **3** integrantes del grupo.

La fecha límite de entrega es el Lunes 12/09/2011 a las 23:59 GMT-0300