DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

# Scheduling

Sistemas Operativos
Segundo Cuatrimestre de 2015


| Integrante | LU | Correo electrónico |
|---|---|---|
| Confalonieri, Gisela Belén | 511/11 | gise_5291@yahoo.com.ar |
| Mignanelli, Alejandro Rubén | 609/11 | minga_titere@hotmail.com |
| Sabarros, Ian | 661/11 | iansden@live.com |





Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

# Índice

# 1. Ejercicio 1 - TaskConsola

Se programó un tipo de tarea llamado `TaskConsola`, que se ocupa de realizar *n* llamadas bloqueantes, cada una con una duración al azar comprendida entre los valores *bmin* y *bmax*.

## 1.1. Algoritmo

La Figura 1 muestra el pseudocódigo de esta tarea.

TASKCONSOLA$(n, bmin, bmax)$

1 **for** $i \leftarrow 0..n$
2 $\quad ciclos\_bloqueo \leftarrow$ valor "random" en $[bmin, bmax]$
3 $\quad$ bloquear durante $ciclos\_bloqueo$ ciclos

Figura 1: Pseudocódigo TaskConsola

Para el cálculo del valor "random" se utilizó la función `rand()` provista por la librería `stdlib` de C++.

# 2. Ejercicio 2 - El lote de Rolando

Se escribió un lote de tareas para simular la siguiente situación:

- Correr un algoritmo que hace un uso intensivo de la CPU por 100 ciclos sin realizar llamadas bloqueantes.
- Escuchar una canción, que realiza 20 llamadas bloqueantes con una duración variable entre 2 y 4 ciclos.
- Navegar por internet, que realiza 25 llamadas bloqueantes con una duración variable entre 2 y 4 ciclos.

Estas tareas se ejecutan simultáneamente.

## 2.1. Pruebas

Se ejecutó el simulador utilizando el algoritmo FCFS para 1 y 2 núcleos, con un cambio de contexto de 4 ciclos. Las Figuras 2 y 3 muestran el resultado de ambas simulaciones.

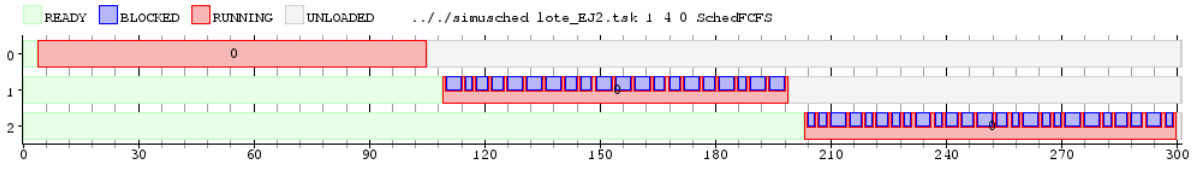


Figura 2: Simulación lote de Rolando con FCFS, 1 núcleo, 4 ciclos de cs

En el caso de la Figura 2, la latencia de cada proceso es:

- **Tarea 0:** latencia 4
- **Tarea 1:** latencia 109
- **Tarea 2:** latencia 202

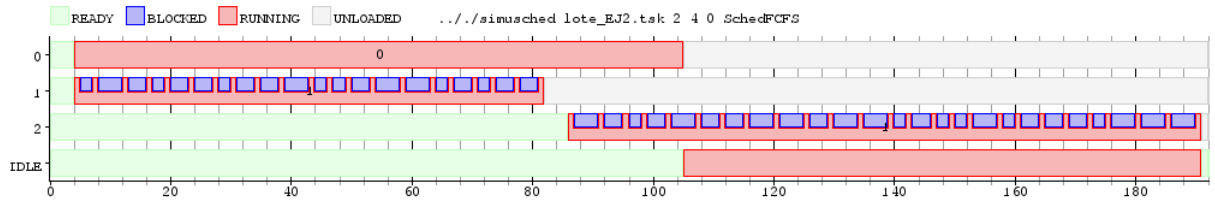


Figura 3: Simulación lote de Rolando con FCFS, 2 núcleos, 4 ciclos de cs

Estos valores nos dan un promedio de 105.
Por otro lado, en el caso de la Figura 3, la latencia de cada proceso es:

- **Tarea 0:** latencia 5
- **Tarea 1:** latencia 5
- **Tarea 2:** latencia 83

Estos valores nos dan un promedio de 31.
Por lo tanto, en el caso en el que Rolando se viera obligado a utilizar una computadora con un solo núcleo... [ESCRIBA AQUÍ SU CONCLUSIÓN]

# 3. Ejercicio 3 - TaskBatch

Se programó un tipo de tarea llamado `TaskBatch`. Este tipo de tarea realiza *cant_bloqueos* llamadas bloqueantes en momentos elegidos pseudoaleatoriamente, y cada bloqueo dura 1 ciclo. Además, utiliza el el CPU durante *total_cpu* ciclos, incluyendo el tiempo necesario para lanzar las llamadas bloqueantes, pero no el tiempo en el que el proceso permanece bloqueado.

## 3.1. Algoritmo

La idea de nuestro algoritmo se basa en decidir, a cada ciclo y de manera pseudoaleatoria, si se realiza un bloqueo o no. Para tomar esta decisión vamos a tomar un valor entero “random” entre 0 y 1 (1 para bloquear y 0 para no hacerlo), nuevamente utilizando la función `rand()` de C++.

Como cada llamada bloqueante consume 1 ciclo de utilización de CPU, podemos decir que *total_cpu* incluye *cant_bloqueos* ciclos destinados a las llamadas bloqueantes, y el resto son usos “puros” de CPU. Por este motivo, la decisión pseudoaleatoria de bloquear la tarea se realizará $total\_cpu - cant\_bloqueos$ veces.

EXPLICAR BIEN POR QUÉ ESTAMOS HACIENDO ESTO
La Figura 4 muestra el pseudo-códgo de este algoritmo.

```
TASKBATCH(total_cpu, cant_bloqueos)
1  for i ← 0..(total_cpu − cant_bloqueos − 1)
2        bloquear ← valor "random" en [0,1]
3        if bloquear == 1∧ aún hay bloqueos por hacer
4              bloquear durante 1 ciclo
5              decrementar cant_bloqueos
6        else usar CPU durante 1 ciclo
7  while aún hay bloqueos por hacer
8        bloquear durante 1 ciclo
9        decrementar cant_bloqueos
```

Figura 4: Pseudocódigo TaskBatch

### 3.2. Pruebas

## 4. Ejercicio 4

## 5. Ejercicio 5

## 6. Ejercicio 6

## 7. Ejercicio 7

## 8. Ejercicio 8