| Nº Orden | Apellido y nombre | L.U. | Cantidad de hojas |
|---|---|---|---|
| | | | |

# Sistemas Operativos
# Primer recuperatorio
# 5 de diciembre de 2013

### Aclaraciones

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Realice cada ejercicio en hojas separadas y NO utilice ambas carillas de la hoja en las respuestas a los ejercicios de semáforos.
- Cada ejercicio se califica con Bien, Regular o Mal. La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se califican globalmente. El parcial se debe aprobar con 2 ejercicios bien y a lo sumo 1 mal.
- El parcial NO es a libro abierto; sin embargo, se permite tener dos hojas A4 con apuntes.
- Por favor entregar esta hoja junto al examen.
- **Importante:** Justifique sus respuestas.

**Ejercicio 1)** Se desea resolver la multiplicación de dos matrices cuadradas $A \times B$ de grandes dimensiones en un procesador que cuenta con cuatro núcleos de procesamiento. Se propone entonces dividir las matriz en cuatro bloques y utilizar un proceso distinto para resolver cada multiplicación de dos submatrices. Luego se deben utilizan los resultados parciales para obtener el resultado final, teniendo en cuenta que:

$$A \times B = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} & A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2} \\ A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1} & A_{1,1} \times B_{1,2} + A_{2,2} \times B_{2,2} \end{bmatrix}$$

Realice el pseudo-código que implemente el algoritmo pedido. Puede asumir que las dimensiones de las matrices son pares.

---

### Solución ejercicio 1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

typedef struct work {
    int p1,p2,p3,p4;
} work;

typedef struct matrix {
    int rows,cols;
    double ** data;
} matrix;

int main(){
    matrix * A, * B;

    A = read_matrix();
    B = read_matrix();
```

```
// Crear pipes
int pipes[4][2];
for(int i = 0; i < 4; i++){
    if(pipe(pipes[i]) < 0){
        perror("creating pipes");
        return 1;
    }
}

// Construir tareas para resolver.
// Indexamos las submatrices asi:
//  A = [ 0, 1,
//        2, 3 ];
//  Ejemplo: A11 es el indice 0 de A, A12 el indice 1.
work tasks[] = {
    // A11 * B11 + A12 * B21
    { 0, 0, 1, 2 },
    // A11 * B12 + A12 * B22
    { 0, 1, 1, 3 },
    // A21 * B11 + A22 * B21
    { 2, 0, 3, 2 },
    // A11 * B12 + A22 * B22
    { 0, 1, 3, 3 },
};

for(int i = 0; i < 4; i++){
    int pid = fork();
    if(pid < 0){
        perror("forking workers");
        return 1;
    }

    if(pid == 0){
        // Soy el hijo.
        // Cerrar pipe de lectura
        if(!close(pipes[i][0])){
            perror("closing read pipe on worker process");
            return 1;
        }

        // Hacer el trabajo. Asumimos:
        // add: suma dos matrices.
        // mult: multiplica dos submatrices, dados los indices de submatriz.
        matrix * res = add( mult(A,B,tasks[i].p1,tasks[i].p2),
                            mult(A,B,tasks[i].p3,tasks[i].p4));

        // Escribir el resultado al pipe. Tambien se podria dejar como
        // pseudocodigo pero lo detallo para que quede claro
        for(int r = 0; r < res->rows; ++r){
            for(int c = 0; c < res->cols; ++c){
                if(write(pipes[i][1],&res->data[r][c],sizeof(double)) < 0){
                    perror("writing result to parent");
                    return 1;
                }
            }
```

```
            }

            // El hijo ya termino, que exitee.
            exit(0);
        }else{
            // Soy el padre. Cerrar pipe de lectura.
            if(!close(pipes[i][1])){
                perror("closing write pipe on parent process");
                return 1;
            }
        }
    }

    // Espero a mis hijos
    for(int i = 0; i < 4; i++){
        int status;
        if(wait(&status) < 0){
            perror("waiting for children");
            return 1;
        }

        if(status != 0){
            fputs("error on child", stderr);
            return 1;
        }
    }

    // Leo el resultado
    matrix * res = build_matrix(rows(A),cols(B));
    for(int i = 0; i < 4; i++){
        // Leer las submatrices en orden. Perdemos paralelismo pero esto se puede
        // resolver usando un select en los 4 pipes. No detallo esta operacion
        // porque es analoga a escribir la matriz.
        read_submatrix_from_pipe(pipes[i][0],res, i);
    }

    // No hay necesidad de liberar nada: Cuando terminemos el SO limpia todo.
    return 0;
}
```

---

**Ejercicio 2)** Para el lote de procesos presentado en la siguiente tabla,

| Proceso | Tiempo de Procesamiento | Instante de llegada |
|---|---|---|
| 1 | 20 | 2 |
| 2 | 14 | 5 |
| 3 | 19 | 7 |
| 4 | 4 | 8 |

a) Calcular el *waiting time*, el *turnaround* y el *response time* promedios para cada una de las políticas de planificación (*scheduling*) indicadas para los casos: a) un sólo núcleo y b) dos núcleos de procesamiento (con un costo de cambio de núcleo de procesamiento de 2 unidades de tiempo).

- FCFS
- RR (quantum=5), con una única cola global, permitiendo así la migración de los procesos entre los dos núcleos (con un costo de cambio de contexto de 1 unidad de tiempo).

Desarrolle su respuesta. Opcionalmente, puede realizar los diagramas de Gantt que considere necesarios para justificar los cálculos realizados.

b) ¿Cuál de los algoritmos elegiría para ser utilizado en un teléfono celular? Justifique.

---

### Solución ejercicio 2

Primero las siguientes definiciones:

- **Waiting Time**: Tiempo que un proceso esta encolado.
- **Response Time**: Tiempo desde que un proceso llega hasta que se ejecute por primera vez.
- **Turnaround**: Tiempo desde que un proceso llega hasta que termina de ejecutarse.

Y las siguientes asumciones:

- Hay un ciclo de espera adicional hasta que se carga el proceso cuando se llega (correspondiente a overhead de inicializar el scheduler).
- Hay un ciclo adicional de procesamiento para cada proceso, en el que este realiza un exit y libera la libc y otros recursos.

En base a estas dos aclaraciones y asumciones, tenemos los siguientes diagramas de GANTT:

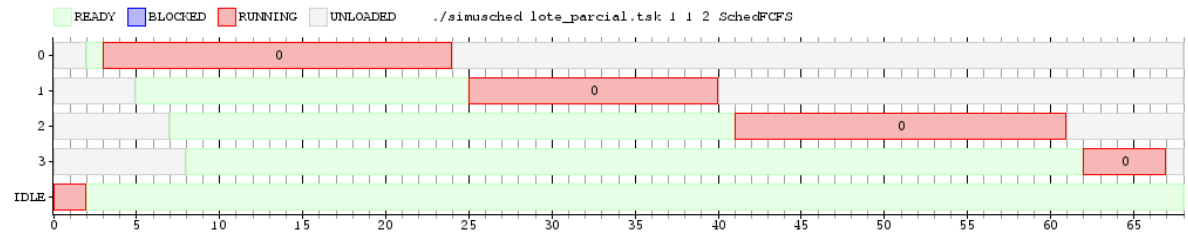


Figura 1: FCFS para un solo core, para el lote de tareas del parcial


Figura 2: FCFS para dos cores, para el lote de tareas del parcial

Para calcular los valores promedio, tenemos que calcular para cada proceso y luego tomar el valor promedio.

Como ejemplo hagamoslo completo para el FCFS con un core. La respuesta a los demás esta incluida y queda como ejercicio. La suma se hace en orden de procesos (e.g. el waiting time del primero 1, del segundo es 20, etc.).

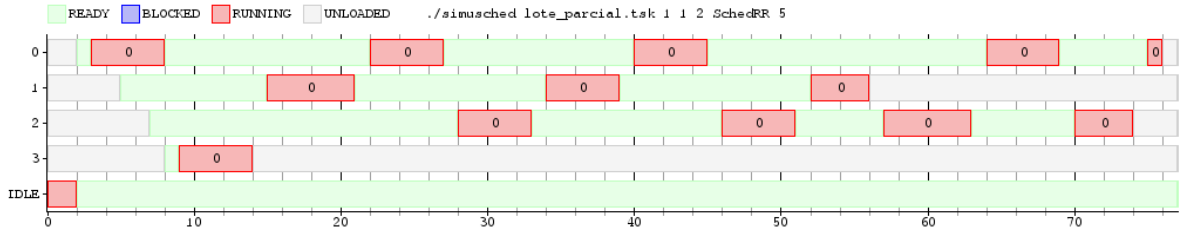


Figura 3: RR para un core, para el lote de tareas del parcial

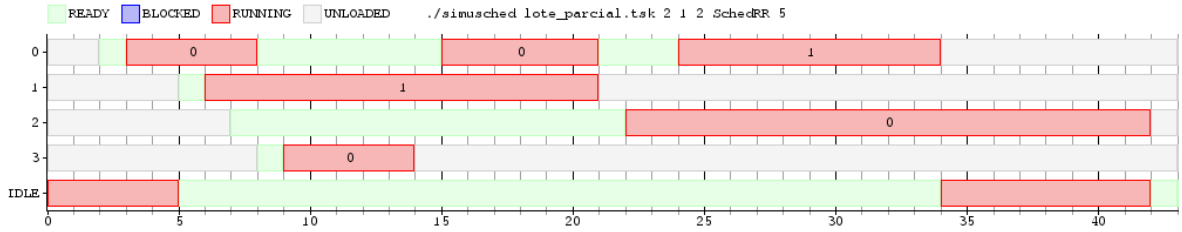


Figura 4: RR para dos cores, para el lote de tareas del parcial

$$waiting\_time = \frac{1 + 20 + 34 + 55}{4} = 27{,}5$$
$$response\_time = \frac{1 + 20 + 34 + 55}{4} = 27{,}5$$
$$turnaround = \frac{22 + 35 + 54 + 59}{4} = 42{,}75$$

Las respuestas para los demas:

- FCFS para 2 cores:
  - Waiting time: 9.000000
  - Response time: 9.000000
  - Turnaround: 24.250000
- RR para 1 core:
  - Waiting time: 36.250000
  - Response time: 5.750000
  - Turnaround: 51.500000
- RR para 3 cores:
  - Waiting time: 13.500000
  - Response time: 2.000000
  - Turnaround: 28.500000

En respuesta al item 2, en un celular nos interesan varias cosas, como bajo uso de bateria y buena velocidad de respuesta para aplicaciones interactivas. En base a esto y al experimento realizado anteriormente, vemos que nos conviene un algoritmo de scheduing con buen response time y para eso nos conviene utilizar un Round Robin

**Ejercicio 3)** Se tienen los siguientes dos procesos, `foo` y `bar` que son ejecutados concurrentemente. Además comparten los semáforos S y R, ambos inicializados en 1, y una variable global x, inicializada en 0.

```
void foo( ) {
    do {
        semWait(S);
        semWait(R);
        x++;
        semSignal(S);
        SemSignal(R);
    } while (1);
}
```

```
void bar( ) {
    do {
        semWait(R);
        semWait(S);
        x--;
        semSignal(S);
        SemSignal(R);
    } while (1);
}
```

a) ¿Puede alguna ejecución de estos procesos terminar en *deadlock*? En caso afirmativo, describir la secuencia y mostrar que se cumplen las 4 condiciones de Coffman.
b) ¿Puede alguna ejecución de estos procesos generar inanición para alguno de los procesos? En caso afirmativo, describir la secuencia.

---

### Solución ejercicio 3

a) Si. La siguiente ejecución genera un deadlock:

- foo ejecuta semWait(S), adquire el semaforo S. scheduler desaloja.
- bar ejecuta semWait(R), adquire el semaforo R. scheduler desaloja.
- foo hace semWait(R), quiere el semaforo R, lo tiene bar.
- bar hace semWait(S), quiere el semaforo S, lo tiene foo.

Veamos que se cumplen las 4 condiciones de Coffman:

- No preemption: Se asume por el scheduler.
- Hold and Wait: foo tiene el recurso S y quiere el recurso R, por ejemplo.
- Exclusion Mutua: Los semaforos empiezan en 1, y un semaforo en 0 no puede ser adquirido. Por lo tanto, al momento de realizarse la secuencia mostrada anteriormente, se puede o no tener un recurso.
- Espera circular: foo tiene S, quiere R, bar tiene R, quiere S, hay un ciclo en el grafo de recursos.

b) No. Probemos asumiendo un quantum de valores entre 1 y 4 (no es necesario otro valor puesto que valores superiores solo hacen que un proceso ejecute toda su seccion critica y vuelva a empezar en el mismo estado).

Dado que los procesos son simetricos con respecto a secciones criticas, podemos considerar que se empieza en cualquiera de ellos. Consideramos que un deadlock no es inanición.

- Con quantum 1, tenemos el deadlock presentado arriba.
- Con quantum 2, bar ejecuta su sección crítica.
- Con quantum 3, tenemos el deadlock presentado arriba.
- Con quantum 4, tenemos el deadlock presentado arriba.

Por lo tanto, no es posible tener inanición asumiendo un scheduler no sesgado con un quantum fijo.

**Ejercicio 4)** En una fábrica de automóviles, una sección de la línea de producción consiste en el ensamblado de dos piezas soldadas que se adozan al auto en construcción. Se cuenta con un brazo soldador que se encarga de armar la carrocería uniendo las partes preensambladas delantera y trasera. El brazo soldador tiene un programa a seguir y cuando éste termina se detiene.

Las partes se encuentran en dos pilas diferentes y son alcanzadas y ubicadas por dos robots (cada uno asignado a una única pila) a la zona de soldadura. Cada robot espera que la zona de soldadura esté libre y existan partes preensambladas en su pila para llevarla a soldar. Cuando las tres partes están en su lugar comienza la soldadura de las mismas. Al finalizar la soldadura de los dos brazos, la carrocería es llevada por una cinta transportadora a la zona de pintura donde es sumergida en una pileta. En esta pileta no puede haber más de dos autos a la vez.

Una vez pintados los autos, uno de los M conductores lleva el auto al estacionamiento (`estacionarAuto()`). Estos deben ser estacionados en el mismo orden en que se pintaron. Luego, los N inspectores proceden a verificar la pintura y el ensamblado del automóvil (`verificarAuto()`), para finalmente entregar en mano (`entregarInforme()`) al único capataz de inspección de la planta quien lo recibe en su oficina (`recibirInforme()`)

Se cuenta con las siguientes funciones ya implementadas:

- `ubicarPiezaDelantera()`: indica al robot que ubique la pieza delantera en la posición correcta.
- `ubicarPiezaTrasera()`: indica al robot que ubique la pieza trasera en la posición correcta.
- `soldadura(posición)`: indica al brazo soldador realizar una soldadura. El parámetro puede ser `TRASERA` o `DELANTERA`.
- `traerAuto()`: indica a la cinta transportadora que traiga un nuevo auto.
- `pintarAuto()`: indica a la cinta transportadora que la soldadura ha terminado y puede llevar el auto a pintar.

Se pide:

a) Utilizando semáforos, escribir los algoritmos en pseudo-código de la cinta transportadora, los dos robots y de el brazo soldador. En el caso de que existan componentes externos que realicen actividades relacionadas a las pilas de partes, explique que operaciones deberán realizar para el correcto funcionamiento del sistema.
b) Explicar brevemente cómo funciona la solución indicando qué semáforos necesitó y para qué.

---

### Solución ejercicio 4

```
Inicializacion:

    zonaSoldadura: 1
    piezasDelanteras: 1
    piezasTraseras: 1
    pilaDelanteras: 1
    pilaTraseras: 1
    piezaDelanteraEnPila: 0
        pero lo tiene que signalear un externo (cuando cargue la pila)
    piezaTraseraEnPila: 0
        pero lo tiene que signalear un externo (cuando cargue la pila)
    hayPiezaDelantera: 0
    hayPiezaTrasera: 0
    lugarParaPintar: 2
    autoParaPintar: 0

Robot de piezas delanteras:

    zonaSoldadura.wait()
    piezasDelanteraEnPila.wait()
    pilaDelanteras.wait()
```

```
    ubicarPiezaDelantera()
    pilaDelanteras.signal()
    zonaSoldadura.signal()
    hayPiezaDelantera.signal()

Robot de piezas traseras:

    zonaSoldadura.wait()
    piezasTraserasEnPila.wait()
    pilaTraseras.wait()
    ubicarPiezaTrasera()
    pilaTraseras.signal()
    zonaSoldadura.signal()
    hayPiezaTrasera.signal()

Brazo Soldador:

    hayPiezaDelantera.wait()
    hayPiezaTrasera.wait()
    zonaSoldadura.wait()
    soldadura(TRASERA)
    soldadura(DELANTERA)
    zonaSoldadura.signal()
    lugarParaPintar.wait()
    pintarAuto()
    autoParaPintar.signal()

Cinta:

    autoParaPintar.wait()
    traerAuto()
    lugarParaPintar.signal()
```

Para el funcionamiento de este sistema, necesitamos un componente externo que ubique las piezas en las pilas. Por ejemplo, para la pila derecha:

```
    pilaDelanteras.wait()
    agregarNuevaPiezaDelantera()
    piezasDelanterasEnPila.signal()
    pilaDelanteras.signal()
```