Primer parcial – 11/10 - 2do. cuatrimestre de 2018

1	2	3	4	Nota
B	R	B	B	A

ACLARACIONES: 1) **Numere** las hojas **entregadas**. Esta hoja se entrega y es la hoja cero. Complete en la primera hoja la cantidad **total de hojas entregadas** (sin contar el enunciado). 2) Realice cada ejercicio en **hojas separadas** y escriba **nombre, apellido y L.U. en cada una**. 3) 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 aprueba con 2 ejercicios bien y a lo sumo 1 mal/incompleto. 4) El parcial **NO** es a libro **abierto**. 5) **Justifique adecuadamente cada una de sus respuestas.**

Ejercicio 1.

Un palíndromo es una palabra o expresión que dice lo mismo si se lee de izquierda a derecha, o de derecha a izquierda. Construya un programa que usando pipes siga la siguiente secuencia: Un proceso le envía una cadena de caracteres recibida por parámetro a otro. Luego, este la invierte y le envía esta nueva versión a un tercer proceso. Este último proceso también recibe la cadena original del primer proceso, las compara, decide si es o no un palíndromo e imprime el resultado por pantalla.

Ejercicio 2.

Considere un algoritmo de scheduling *preemptive* y con prioridades dinámicas, dónde números mayores implican mayor prioridad. A todos los procesos entrantes se les asigna inicialmente una prioridad 1. Cuando un proceso está esperando por la CPU (en la cola de preparados, pero no ejecutándose), su prioridad cambia a una tasa de α. Cuando se está ejecutando, su prioridad cambia a una tasa de β. En resumen, si $P_t(i)$ representa la prioridad del proceso i en el tiempo t:

$$P_{t+1}(i) = \begin{cases} P_t(i) + \alpha * P_t(i) & \text{si el proceso i está en espera} \\ P_t(i) + \beta * P_t(i) & \text{si el proceso i está ejecutando} \end{cases}$$

Los parámetros α y β se pueden configurar para muchos tipos diferentes de algoritmos de scheduling. Notar que la prioridad de un proceso es siempre mayor a cero. Justificando su respuesta, explique:

a) Describir el comportamiento que resulta al tener $1 > \beta > \alpha > 0$.
b) Describir el comportamiento que resulta al tener $-1 < \beta < \alpha < 0$.

A responder, describiendo las diferencias entre los procesos tratados por CPU y por Entrada/Salida.

Ejercicio 3.

Se desea implementar un sistema de generación de casos de test para aplicaciones móviles. Para tal fin, se cuenta con n procesos **Celular** y un proceso **Generador**. Implemente el código necesario para el proceso Generador y Celular de forma que cumpla lo pedido a continuación. No se olvide de dejar bien en claro cuales son las variables compartidas. Agregue comentarios donde considere necesario para esclarecer la idea de la implementación.

En primer lugar, el proceso Generador compila el código fuente de la aplicación mediante la función `string compilarAplicación()`. Esta función devuelve como resultado la ubicación del archivo compilado.

Luego, cada proceso Celular instala la aplicación compilada mediante la función `bool instalarAplicacion(int id, string ubicacion)`, que toma como primer parámetro el número del celular (puede asumir que cada proceso Celular conoce su id) y cómo segundo parámetro la ubicación del archivo compilado. Esta función devuelve verdadero en caso de que la instalación haya sido exitosa y falso en caso contrario. Si la instalación en un Celular falla, se debe abortar todo el proceso de generación.

Una vez que todos los Celulares terminaron de instalar la aplicación, el proceso Generador determina la longitud de cada uno de los casos de test que se generarán utilizando la función `int randomInt()`. Se puede asumir que existe una constante `TOTAL` (mayor que n) que indica la cantidad deseada de casos de test, a ser generados en conjunto por todos los procesos Celular. Cada uno de los casos de test deberá ser generado por un único proceso Celular. *A medida que* la longitud de cada caso de test sea determinada, cada proceso Celular podrá empezar a generar casos de test utilizando la función `string generarTest(int longitud)` que devuelve como resultado el caso de test generado. Para esta parte, no está permitido asumir de manera previa qué Celular generará cada uno de los casos de test.

Para finalizar, el proceso Generador escribe todos los casos de test a disco utilizando la función `void escribirTests(list<string> tests)` que toma como parámetro los casos de test generados.

Ejercicio 4.

Se tiene un sistema con 8 páginas y sólo 4 marcos de página. La memoria comienza vacía. Llegan los siguientes pedidos de memoria (número de página) en el siguiente orden: 5, 2, 3, 5, 2, 7, 5, 2, 6, 4.

Indique qué página se desaloja tras cada pedido utilizando los algoritmos FIFO, LRU y Second Chance y calcule el *hit-rate* en cada caso.

Corrector: Rodolfo Suñoga

1) PALINDROMO (STRING PALABRA) {

```
Pipe_Fd1 = Pipe()
Pipe_Fd2 = Pipe()
Pipe_Fd3 = Pipe()
PID = FORK()
IF ( Pid == 0) {  // Soy HIJO, PRIMER PROCESO
    CLOSE( PIPE_Fd1[0])
    dup2 (PIPE_Fd1[1], STD_OUT)  // PIPE_Fd1 lo tomo como salida
                                 // Se obtendrá PALABRA del otro LADO.
    PRINT ( PALABRA)
}else{ // Soy PADRE
    PID = FORK()
    IF (PID == 0) {  // Soy HIJO, Segundo Proceso
        CLOSE(PIPE_Fd1[1])
        CLOSE (PIPE_Fd2[0])
        dup2 ( Pipe_Fd1[0], STd_iN)  // Tomo PIPE_Fd1 como ENTRADA
        dup2 ( Pipe_Fd[1], STd_OUT)  // Tomo PIPE_Fd[ como SALIDA
        PALABRA_INVERTIDA = INVERTIR ( PALABRA)  // LA INVIERTO
        PRINT (PALABRA_INVERTIDA)  // ENVIO LA PALABRA INVERTIDA POR PIPE_Fd2
    }else{ // Soy PADRE
        PID = FORK()
        IF(PID==0) {  // Soy HIJO, TERCER PROCESO
            CLOSE( PIPE_Fd1[1])
            CLOSE( PIPE_Fd2[1])
            dup2( Pipe_Fd1[0], STD_IN)  // Tomo AMBOS
            dup2( Pipe_Fd2[0], STD_IN)  // como ENTRADA
            dup2 ( PIPE_Fd3[1], STD_OUT)  // Tomo PIPE3 como SALIDA
            CIN >> PALABRA  // de AMBOS PIPES
            CIN >> PALABRA_INVERTIDA  // TOMO LAS PALABRAS
            IF ( PALABRA == PALABRA_INVERTIDA) {
                PRINT ("ES PALINDROMO")  // ENVIO RESULTADO
            }else{
                PRINT ("NO ES PALINDROMO")  // POR PIPE 3
            }
        }
    }
}
```

Xx estas registrando todos los extremos y salidas en pipes

Es mejor escribir directo a pipe

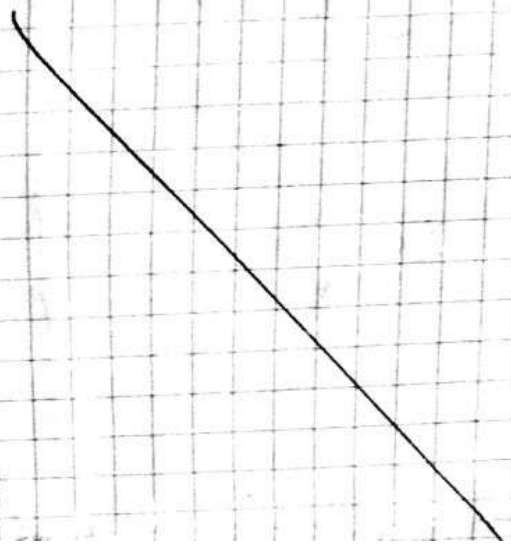
CIN >> PALABRA

Asignamos la palabra invertida.

Close(PIPE_Fd2[0])

Scanned by CamScanner

```
Else{  // Soy tu PADRE wait :
    Close(Pipe_Fd3[1])
    dup2(Pipe_Fd3[0], Std_IN)  // TOMO Pipe 3 como STD IN
    Cin >> Resultado  // OBTENGO RESULTADO
    PRINT(Resultado)  // IMPRIMO EN PANTALLA
}
```



2. Al ser un algoritmo preemptivo se sabe que al momento que el ~~procesador~~ proceso se bloquea por la espera de E/S, el procesador pasará a ejecutar el siguiente proceso con mayor prioridad.

a) En el caso en donde $0 < \alpha < \beta < 1$

ocurrirá que un proceso acotado por CPU ejecutará mas tiempo que uno acotado por E/S, ya que al ser $\beta > \alpha$, la prioridad del proceso acotado por CPU incrementará de una manera más rápida que el E/S. Por lo tanto ~~[illegible]~~ los procesos acotados por ~~est~~ CPU tendrán siempre la ventaja de prioridades, pero como siempre se le suman prioridades, nunca ocurrirá que un proceso no corra (STARVING). ~~[illegible]~~ !!

b) En este caso $-1 < \beta < \alpha < 0$

Por lo tanto las prioridades irán disminuyendo, ya que las ~~[illegible]~~ son negativas. Luego, observamos que los procesos acotados por E/S disminuirán mas lento que los acotados por CPU, lo que nos llevará a obtener que los procesos acotados por E/S ~~[illegible]~~ finalizarán en primera instancia? No!!

El gran problema en este caso, es que puede ocurrir starving. Ya que los procesos siempre disminuirán de prioridad, y ~~[illegible]~~ puede ocurrir que se agreguen nuevos procesos, los cuales tendrán prioridad 1, el cual siempre será mayor que los demas.

3)

VARIABLES GLOBALES:

```
UBICACION = " "
COMPILADO = SEM(0)
ATOMIC<INT> CELULARES_INSTALADOS = 0
INSTALACIONES_COMPLETAS = MUTEX(0)
INSTALADAS_EXITOSAMENTE = FALSE
INT LONGITUDES[TOTAL]
LONGITUDES_OBTENIDAS[TOTAL] = [MUTEX(0) FOR i IN RANGE(0,TOTAL)]
ATOMIC<INT> INDICE_TEST = 0
RES_TESTS = []
CELULARES_CONTINUEN = SEM(0)

GENERADOR() {
    UBICACION = COMPILAR_APLICACION()
    FOR(i IN RANGE(0,N)) { COMPILADO.SIGNAL() } // AVISO A LOS CELULARES QUE YA ESTA COMPILADO
    INSTALACIONES_COMPLETAS.WAIT() // ESPERO QUE LOS CELULARES TERMINEN DE INSTALAR
    FOR(i IN RANGE(0,N)) { CELULARES_CONTINUEN.SIGNAL() } // AVISO QUE ME ENTERE QUE FINALIZARON
    IF(!INSTALADAS_EXITOSAMENTE) { EXIT() } // CHEQUEO SI HUBO ERRORES.
    FOR(i IN RANGE(0,TOTAL)) {
        LONGITUDES[i] = RANDOM.INT()   // OBTENGO LONGITUDES
        LONGITUDES_OBTENIDAS[i].SIGNAL()   // AVISO QUE EL i-ESIMO TEST PUEDE COMENZAR
    }
    FOR(i IN RANGE(0,TOTAL)) { TESTS_FINALIZADOS.WAIT() } // ESPERO QUE TODOS LOS TESTS HAYAN TERMINADO
    TEST(RES_TESTS)
}
```

No funciona, puede haber un error sin ser detectado.

Scanned by CamScanner

```
CELULAR (ID) {
  COMPILADO.WAIT()  // ESPERO QUE EL GENERADOR COMPILE EL TEST
  LOGRADO = INSTALAR APLICACION(ID, UBICACION)
  IF(!LOGRADO){ INSTALADAS_EXITOSAMENTE = FALSE; // SETEO QUE HUBO UN PROBLEMA
  CELULARES_INSTALADOS.GETANDADD(1) // SUMO 1 AL TOTAL DE CELULARES INSTALADOS
  IF (CELULARES_INSTALADOS.GET() == N) { INSTALACIONES_COMPLETAS.SIGNAL()}
  CELULARES_CONTINUEN.WAIT() // ESPERO QUE EL GENERADOR ME AVISE QUE TODOS LOS CELULARES TERMINAMOS
  IF(!INSTALADAS_EXITOSAMENTE) { EXIT()} // SI HUBO ERROR ABORTO
  INDICE = INDICE_TEST.GETANDADD() // AL SER ATOMICO SE QUE VOY A SER EL UNICO CELULAR CORRIENDO EL TEST
  WHILE (INDICE < TOTAL)
      LONGITUDES_OBTENIDAS.WAIT() // ESPERO QUE EL GENERADOR OBTENGA LA LONGITUD
      RESTESTS.PUSH_BACK(GENERAR_TEST(LONGITUDES[INDICE])) // AGREGO EL RESULTADO
      TESTS_FINALIZADOS.SIGNAL() // AVISO QUE HAY UN TEST MENOS QUE HACER
      INDICE = INDICE_TEST.GETANDADD(1) // OBTENGO EL PRÓXIMO INDICE
}
```

SOY EL n-ESIMO CELULAR QUE TERMINO DE INSTALAR → AVISO

NOTA: ESTO Y SUPONIENDO QUE N ES UN VALOR DADO Y QUE ES LA CANTIDAD DE PROCESOS CELULAR.

4)

MARCOS

A
B
C
D

PÁGINAS

1
2
3
4
5
6
7
8

FIFO:

F FIRST
I IN
F FIRST
O OUT

El primero que entra es el primero que sale

PEDIDO	MARCOS A	B	C	D	HIT
3	X	X	X	X	X
2	3	X	X	X	X
3	3	2	X	X	✓
5	3	2	X	X	X
2	3	2	5	X	✓
7	3	2	5	X	X
8	~~3~~	2	5	7	X
2	8	~~2~~	5	7	✓
6	8	2	~~5~~	7	X
4	8	6	5	~~7~~	X
FINAL	8	6	5	4	

HIT RATE

$\frac{3}{10}$ ✓

LRU:

L LEAST
R RECENTLY
U USED

Sale el que hace mas tiempo no se usa.

PEDIDO	MARCOS A	B	C	D	HIT
3	X	X	X	X	X
2	3	X	X	X	X
3	3	2	X	X	✓
5	3	2	X	X	X
2	3	2	5	X	✓
7	3	2	5	X	X
8	~~3~~	2	5	7	X
2	8	2	5	7	✓
6	8	2	~~5~~	7	X
4	8	2	6	~~7~~	X
FINAL	8	2	6	4	

HIT RATE:

$\frac{3}{10}$ ✓

Second-Chance:

Funciona como uno FIFO ✓ pero cuando se utiliza la página se le da una segunda chance.

MARCOS

PEDIDO	A	B	C	D	HIT
3	X	X	X	X	X
1	3'	X	X	X	X
3	3'	2'	X	X	✓
5	3	2	1	X	X
2	3	2	5'	X	✓
7	3	2	5	X	X
8	3	2	5	7	X
2	3	2	8	7	✓
6	3	2	8	7	X
4	3	2	8	6	X
FINAL	4	2	8	6	

HIT RATE

$\frac{3}{10}$ ✓

✓

ESTADOS FINALES:

MÉTODO	A	B	C	D	HIT RATE
FIFO	8	6	5	4	0,3
LRU	8	2	6	4	0,3
SC	4	2	8	6	0,3

✓ ✓ ✓

Se observa que todos dieron el mismo HIT RATE, pero ninguno terminó con las páginas en el mismo lugar.