# SISTEMAS OPERATIVOS: INFORME PROYECTO N°1


**INTEGRANTES:**

Ceballo Vitale, Pablo Guillermo

Gómez, Tomás Alejandro

**Segundo Cuatrimestre de 2018**

**Fecha y Año: 1/10/2018**

**Profesor a cargo: Cenci, Karina**

# EJECUCIÓN DE LOS PROGRAMAS

Para ejecutar los programas asociados a este proyecto, abra un terminal de su preferencia y diríjase hasta la carpeta “Codigo C” por medio del comando “cd”, luego ejecute el script asociado al programa que desee ejecutar por medio del comando “./<Nombre_del_Script>” estos scripts compilarán y ejecutarán el código con los parámetros necesarios para su correcto funcionamiento.

En caso de que alguno de los scripts falle debido a problemas de permisos, se adjunta con los fuentes y scripts un Script llamado “MAKE_Permisos” el cual al ejecutarse modificará los permisos de todos los scripts de forma tal que puedan ser ejecutados

Los scripts disponibles para ejecutar son:

- MAKE_Asistente
- MAKE_Impresoras
- MAKE_ImpresorasPrioridades
- MAKE_Shell
- MAKE_SudokuProcesos
- MAKE_SudokuThread
- MAKE_Permisos

Para ejecutar los scripts asociados a los Sudokus, deberá agregar un archivo de texto denominado “sudoku.txt” con la grilla de sudoku a verificar

# NOTAS DE DESARROLLO EJERCICIO 1

*COMENTARIO GENERAL*: El reporte de errores se hace a través de salidas por consola por medio de impresiones con printf o fprintf

## SUDOKU

### SUDOKU CON PROCESOS

Hacemos uso de 3 Procesos, uno que se encarga de Verificar las Filas del Sudoku, otro que se encarga de Verificar las Columnas del Sudoku y un proceso que Verificará los 9 Cuadrantes del Sudoku. Estos procesos Escriben los resultados de Verificar la correctitud de su parte de la jugada en un su archivo distinto correspondiente, el cual el Proceso Padre levantará una vez que los hijos terminaron para verificar la correctitud de la jugada.

**Algoritmo General:**

Abrir los archivos “Proc1”, “Proc2” y “Proc3” en modo escritura
Abrir el archivo “sudoku.txt” en modo lectura
si se pudo abrir el archivo correctamente
    Leer el archivo caracter a caracter y llenar la Matriz “GrillaSudoku” con la función “Lectura(...)”
    cerrar el archivo “sudoku.txt”
Si se pudo verificar la Completitud de la Grilla con la función “Completitud(...)”
    para i de 0 a 3 y mientras check1 esté en verdadero
        PID <- “fork()”
        si (PID = -1)
            Reportar la existencia de un error en la bifurcación
        si (PID != 0)
            check1 <- HacerTarea(i)
            exit()
        sino //Estoy en el proceso Padre
            Esperar por todos los Procesos Hijos

abrir los archivos “Proc1”, “Proc2” y “Proc3” en modo lectura
si los archivos se abrieron correctamente
    si el contenido de los tres archivos en simultáneo es la palabra “true”
        Escribir por pantalla “La jugada era valida”
    Sino
        Escribir por pantalla “La jugada NO era válida”

//FIN ALGORITMO GENERAL

```
Función Lectura(FILE *Sud, char Sudoku[][9])
	F <- 0
	C <- 0
	Mientras no haya llegado a Fin de Archivo y F sea menor a 9{
		Mientras C sea menor a 9{
			char num <- un caracter leído del archivo Sud
			Si el caracter NO es EOF o una coma o Fin de Linea o algo que NO
sea un número del 1 al 9
				guardo el valor en Sudoku[F][C]
			Sino
				guardo un NULL en Sudoku[F][C]
		C++
		}
	F++
	C <- 0
	}
//FIN ALGORITMO

Función Completitud(char Sudoku[][9])
	OK <- true
	Para un F entre 0 y 9
		Para un C entre 0 y 9
			si Sudoku[F][C] es NULL
				OK <- False
	retornar el valor de verdad de la variable OK
//FIN ALGORITMO
```

```
Función HacerTarea(int Nro):
	rec <- 0 , Y <- 0
	check <- true
	Dependiendo de Nro{
		Caso Nro = 0
			Mientras check sea verdadero y rec sea menor a 9
				check <- VerificarFila(Sudoku,rec)
				rec++
		Caso Nro = 1
			Mientras check sea verdadero y rec sea menor a 9
				check <- VerificarColumna(Sudoku,rec)
				rec++
		Caso Nro = 2
			Mientras check sea verdadero, rec sea ,menor e igual que 6 e Y sea
menor e igual que 6
				check <- VerificarCuadrante(Sudoku,rec,Y)
				rec <- rec + 3
				si rec = 6 pero Y != 6{
					rec <- 0
					Y <- Y + 3
				}
	}
	devolver el valor de verdad guardado en check
//FIN ALGORITMO

Función VerificarFila(char Sudoku[][9], int F)
	bool Lista[9]
	Para un I entre 0 y 9, I++
		int X <- Sudoku[F][I] - '0'
		si X es un valor que NO esta entre 1 y 9 (incluidos)
			Escribir "false" en el archivo "Proc1"
			devolver falso
		si X es un Valor que YA ESTABA en Lista[I]
			Escribir "false" en el archivo "Proc1"
			devolver falso
		Lista[X] <- true

	Para un I entre 0 y 9, I++
		X <- I + 1
		si Lista[X] = false
			escribir "false" en "Proc1"
			devolver falso

	escribir "true" en "Proc1"
	devolver true
//FIN ALGORITMO
```

El Algoritmo para la Función VerificarColumna(char[][9] Sudoku, int C) es similar al de la Función VerificarFila(...), las únicas diferencias son la linea remarcada en amarillo, la cual es reemplazada por: int X <- Sudoku[F][C] - '0' y que se escribe el resultado de la operación en el archivo "Proc2"

```
Función VerificarCuadrante(char Sudoku[][9], int X, int Y)
        bool Lista[9]
        int I
        int J
        para I = X, I menor estricto que X + 3, I++
                para J = Y , J menor estricto que Y + 3, J++
                        int Z <- Sudoku[I][J] - '0'
                        si Z es un valor que NO esta entre 1 y 9 (incluidos)
                                Escribir "false" en el archivo "Proc3"
                                devolver falso
                        si Z es un Valor que YA ESTABA en Lista[Z]
                                Escribir "false" en el archivo "Proc1"
                                devolver falso
                Lista[Z] <- true

        Para un I entre 0 y 9, I++
                int R <- I + 1
                si Lista[R] = false
                        escribir "false" en "Proc3"
                        devolver falso

        escribir "true" en "Proc3"
        devolver true
//FIN ALGORITMO
```

## SUDOKU CON HILOS

Hacemos uso de 11 Hilos para este problema, uno para verificar las Filas del Sudoku, otro para verificar las Columnas del Sudoku y los 9 restantes se encargarán de Verificar cada cuadrante del Sudoku por separado, los Hilos se comunican con el proceso Padre a través de un vector de valores booleanos, en los cuales se deja constancia del resultado de la verificación. Se hace uso de una Estructura llamada "datos_thread" en la cual se almacenarán los datos de importancia a los efectos de la verificación del Sudoku (Fila/Columna de Inicio y Fila/Columna de Fin) y la Posición del arreglo de valores booleanos a usar para el hilo

```
//Algoritmo General

FILE *SudokuR
char Sudoku[9][9];
int rc
pthread_t Hilos[NUM_THREADS] //NUM_THREADS = 11
Cargar()
SudokuR <- abrir el archivo “sudoku.txt” en modo lectura
Si NO se pudo abrir el archivo
        reportar el error
Sino
        Lectura(&SudokuR,Sudoku)
int i
para i entre 0 y NUM_THREADS
        check[i] <- true //check es un arreglo de valores booleanos de alcance global
int j
para j entre 0 y NUM_THREADS
        rc <- crear el hilo y almacenarlo en el arreglo de Hilos en la posición j, asiganandole
el conjunto de datos Datos_Thread[j] y la función VerificarParte para ejecutar
        si rc
                Reportar que hubo un error al crear el hilo y terminar
//FIN ALGORITMO GENERAL

Función VerificarParte(void *threadarg)

bool nums[10]
struct datos_thread *misDatos
int F
misDatos <- (struct thread_data *) threadarg;
para F <- Fila Inicial a verificar, F menor que Fila Finala verificar por el Hilo, F++
        para C <- Columna Inicial a verificar, C menor que la Columna Final a verificar, C++
                int num <- Sudoku[F][C] - ‘0’
                SI num NO es un dígito del 0 al 9
                        check[pos] = false
                Si nums[num] NO es falso
                        check[pos] = false
                sino
                        nums[num] = true
int i
para i = 1 , i < 10 y check[pos], i++
        si nums[i] NO es true
                check[pos] = false
terminar Ejecución del hilo

//FIN ALGORITMO
```

# SHELL

Para este ejercicio, se asume que el comando a ejecutar SOLO separa la instrucción de los argumentos y los argumentos entre si por medio de espacios.

Este ejercicio hace uso de la librería “strtok” para facilitar la lectura, procesado y ejecución de los comandos ingresados por consola, esta función provee de una forma de leer, dado un token que haga de separador entre las cadenas de caracteres, que se basa en leer la cadena hasta que se llegue al terminador especificado.

Entonces, la estrategia adoptada para este problema es la siguiente: Una vez ingresado el comando a ejecutar, el programa lee la secuencia de caracteres haciendo uso de la función “getline”, guardandola en un puntero, para luego hacer uso de la función “strtok” que sistemáticamente separará el comando de los argumentos del mismo y los guardará por separado en un arreglo, entonces, después de verificar si el comando se puede ejecutar, se bifurca a través del uso de “fork” y el proceso Hijo ejecutará el comando con sus argumentos a través de la llamada a la función “execvp”, tomando el primer lugar del arreglo de argumentos como el nombre del comando a ejecutar y las demás posiciones del arreglo como los argumentos de la función, esto hará que se ejecute el System Call asociado a la función pedida

# EJERCICIOS DE SINCRONIZACIÓN

NOTA: en las explicaciones de los ejercicios de semáforos se usaron los términos “levanta/libera” para referirse a sem_post(sem_t) (es decir, aumentar el valor del semáforo en 1) y bajar para sem_wait(sem_t) (es decir, decrementar el valor del semáforo en 1).

## IMPRESORAS

### SIMPLE

Se utiliza 6 hilos como ejemplo que hacen de los usuarios que usan la impresora, junto con un semáforo ImpresorasDisponibles que indica la cantidad de impresoras actualmente desocupadas (el cual varia entre 2 y 0). En la función main se inicializan tanto el semáforo como los hilos y se manda a cada hilo usuario hacia la función RequerirEImprimir. Esta función maneja la solicitud de una impresora y a la vez el uso de la misma, tal que al llegar, el hilo usuario repite durante 20 ciclos lo siguiente: Solicita una de las 2 impresoras disponibles (si no hay ninguna, espera a que otro usuario termine de usarla), luego realiza la actividad de impresión (en nuestro caso lo simbolizamos con dos printf y un sleep), finalmente, el usuario deja (libera) la impresora disponible para otro usuario. Cabe destacar que cada vez que se usa o libera una impresora, se modifica el valor del semáforo ImpresorasDisponibles. Se utilizó la librería semaphore.h para trabajar con semáforos, siendo cada uno de tipo sem_t. También se usaron hilos de tipo pthread_t, en los cuales, si llegara a fallar la creación de los mismos, se notifica de un error.

```
/*Algoritmo RequerirEImprimir

   //Durante 20 ciclos, hacer
        //Entry
                //Requerir(ImpresorasDisponibles)
                //Mientras no tengo el lock, esperar
        //Sección Crítica, obtuve el lock
                //Hacer trabajo de impresion
        //Exit
                //Liberar(ImpresorasDisponibles)

*/
```

## IMPRESORAS CON ESQUEMA DE PRIORIDADES

Para las impresoras con prioridades se utilizan hilos y semáforos de la misma forma que antes, solamente que esta vez se tiene: el semáforo binario BloquearImpresora usado para evitar que dos impresoras busquen usuarios a la vez, dos arreglos de impresoras, ambos del mismo tamaño que la cantidad de hilos de usuarios (en nuestro caso, 6), estos dos arreglos son Prioridades (el cual es utilizado por las impresoras para ver cuál prioridad mirar) y EsperandoImpresion (usado para que el usuario espere en su lugar mientras la impresora hace su trabajo), ambos inicializados en 0 para todos sus semáforos. Además de los hilos de Usuario se tienen dos hilos de Impresoras, definidos de la misma manera. Fue necesario utilizar una estructura asociada a cada hilo de Usuario que sirvió para mantener en cada uno su prioridad propia, tal que en nuestro caso, asumimos que la prioridad es según su orden de creación (el primer hilo creado tiene prioridad 0, y el segundo prioridad 1), tal que a mayor valor numérico, más alta es la prioridad. Esta estructura es usada tal que, cuando un Usuario quiere manipular su propio semáforo en el arreglo (ya sea Prioridad o EsperandoImpresion), al tener acceso a su prioridad, puede acceder a su semáforo en el arreglo usando ese valor.
A los Usuarios se les asigna la función Requerir, donde primero se obtiene la estructura para cada hilo, y luego, durante 20 ciclos, el Usuario libera su propio semáforo en Prioridades (lo setea en 1) para que una impresora lo detecte y lo tenga en cuenta al buscar por prioridades, luego espera a su propio EsperandoImpresion, hasta que la impresora imprima su pedido y libere ese semáforo para que el Usuario vuelva a bajarlo.
Mientras tanto, se tienen los dos hilos de Impresoras asignados a la función Imprimir, tal que durante 60 ciclos, el hilo comienza queriendo bajar BloquearImpresora para proseguir con la busqueda del Usuario de mayor prioridad que quiere imprimir (es decir, que tiene su semáforo de Prioridad en 1). Para esto, por cada semáforo en Prioridad, la impresora realiza un sem_trywait tal que, si ese Usuario quiere imprimir, lo toma, sino, intenta con el que sigue (este recorrido se realiza de mayor a menor prioridad, es decir, del número más alto al más bajo) (esto se hace tal que, si el valor retornado por el trywait el -1, significa que no se pudo tomar el semáforo, caso contrario, pudo tomarse sin problema). Si no encuentra un Usuario que quiere imprimir al recorrer todo el arreglo, libera BloquearImpresora, tal que le deja realizar una búsqueda a la otra impresora y esta se queda esperando a que termine y libere BloquearImpresora. Si encuentra un Usuario, baja su semáforo en Prioridad y libera el de BloquearImpresora, para que realice su propia búsqueda, luego la impresora realiza el proceso de impresión (en nuestro caso, ejecuta un printf), y luego libera el EsperandoImpresion de ese Usuario de forma tal que le notifica que el trabajo ya terminó. Para terminar setea un booleano encontré en true para que no libere BloquearImpresora innecesariamente.
Al terminar, todos los semáforos se destruyen y los hilos terminan.

```
/*Algoritmo Imprimir

  //Durante 60 ciclos, hacer
      //Requerir(BloquearImpresora) para evitar que ambas impresoras se sobrepongan
      //Encontre<-false
```

```
        //Para cada usuario, comenzando desde la prioridad mas alta, se recupera el primer pedido disponible encontrado y mientras Encontre=false
                //Intentar Requerir(Prioridades)
                        //Si se puede, Liberar(BloquearImpresora) para que la otra busque un pedido
                        //Realizar trabajo impresion
                         //Liberar(EsperandoImpresion) para que el usuario continue su trabajo
                        //Encontre<-true
        //Si no Encontre pedido disponible
                //Liberar(BloquearImpresora) para que la otra impresora realize su busqueda
*/

/*Algoritmo Requerir

   //Durante 20 ciclos, hacer
        //Liberar(Prioridades propia) para entrar en la lista de pedidos
        //Requerir(EsperandoImpresion propia) para esperar a que termine de imprimir
*/
```

# ASISTENTE

Esta vez se tienen 4 semáforos: EsperarTurno, Atendido, Asiento y OficinaLibre; junto con hilos representando los Alumnos (en este caso 3) y un hilo representando al Asistente. Al crear todos los hilos, se tiene que, los Alumnos son asignados a la función Solicitar y el Asistente a Atender.
En Atender, durante 20 ciclos, el Alumno comienza intentando con sem_trywait tomar un Asiento (el cual se inicializa en 3 para representar los 3 asientos disponibles), si puede tomarlo significa que hay un asiento disponible, si la ejecución NO entra a la Sección Crítica, se debe a que los 3 están ocupados. Tanto si está ocupado como si pudo realizar su consulta, debe esperar un sleep(10) antes de volver. Si consigue Asiento lo ocupa (baja el semáforo en uno), y libera el semáforo EsperarTurno (inicializado en 0), tal que se usa para notificar al asistente que hay un Alumno esperando su turno. Luego el Alumno baja el semáforo binario OficinaLibre (inicializado en 1) que representa si hay un alumno o no en la oficina; si puede tomarlo, es que la oficina estaba vacia y puede pasar a realizar su consulta. De esta forma el Alumno libera su Asiento para otra persona y pasa dentro. Luego el Alumno intenta bajar el semáforo binario Atendido (inicializado en 0), el cual simboliza que la consulta ya terminó (lo cual significa que el Alumno solo podrá bajar Atendido una vez que el Asistente termine la consulta y lo levante él mismo). Cuando termine la consulta, el Alumno levanta OficinaLibre para dejarle el paso a uno de los alumnos esperando su consulta y se va.
Por el lado de Atender, se tiene que durante 60 ciclos, el hilo de Asistente se quedará esperando a que un Alumno levante EsperarTurno (esto simboliza que el Asistente está durmiendo). Al levantarse EsperarTurno, el Asistente vuelve a bajarla y responde a la consulta (esto se simboliza con un printf y un sleep(4)). Al terminar, el Asistente levanta Atendido tal que de esta forma, el Alumno puede bajarlo en Solicitar e irse. Luego el Asistente vuelve a esperar otra vez a EsperarTurno.

```
/*Algoritmo Atender

   //Durante 60 ciclos, hacer
        //Esperar un turno de un alumno (wait(EsperarAlumno))
        //Atender alumno
        //Liberar(Atendido)
*/
```

```
/*Algoritmo Solicitar

   //Durante 20 ciclos, hacer
        //Verificar si hay asientos (trywait(Asiento))
        //Si no hay Asientos
                //Irse y esperar un tiempo antes de volver a intentar
        //Si hay un asiento disponible, ocuparlo
                //Pedir turno (liberar(EsperarTurno))
                //El alumno espera a que la oficina este libre para entrar, no deja su asiento
todavia (wait(OficinaLibre))
                //Liberar(Asiento)
                //Esperar a que termine de ser atentido por el asistente (wait(Atendido))
                //El alumno abandona la oficina (Liberar(OficinaLibre))
                //Al terminar, esperar un tiempo antes de volver a consultar
*/
```

# CONSIDERACIONES Y RESOLUCIONES EJERCICIO 2

## RESOLUCIÓN ARTICULO INTEL

a) Hubo un tiempo en el que para cambiar el sistema operativo de una PC era necesario cambiar todo el gabinete de la misma, tal cual me ocurrió en el año 2007. Mi computadora con sistema operativo Windows XP estaba dejando de funcionar, razón por la cual mis padres encargaron a un familiar un nuevo gabinete para PC que no tenía un Sistema Operativo instalado de fábrica, por ello mi padre decidió conseguir e instalarle una versión oficial del Sistema Operativo Windows Vista (nuevo en ese momento). Éste ofrecía una gran variedad de nuevos servicios y funcionalidades y a la vez usaba una gama de colores mucho más llamativa.

En cambio, en la actualidad, el pasaje de un sistema operativo a otro es tan simple como presionar un botón, tal cual me sucedió el día 24 de septiembre del presente año, cuando mi computadora con sistema operativo Windows 10 mostró el siguiente mensaje en la pantalla: “Es necesario reiniciar el dispositivo para instalar las actualizaciones”. Siempre me disgustó sobremanera ese mensaje ya que, en muchas oportunidades, el cambio de OS significaba para mí terminar en reducciones de performance en la computadora. Al presionar el botón de reinicio, el sistema comenzó el proceso de actualización que llevó aproximadamente unos 45 minutos y varios reinicios del dispositivo. Una vez terminada la actualización, el sistema operativo solicitó que ingresara los datos de mi cuenta. Aunque ofrecía algunas opciones de inicio de sesión nuevas, sólo me dispuse a ingresar con la contraseña. Luego de otra espera de aproximadamente 5 minutos, un programa me introdujo a las nuevas funcionalidades que ofrecía esta nueva versión de Windows 10.

Sólo llegué a darme cuenta de que ahora este cambio era distinto luego de haber leído el artículo publicado por Intel sobre la evolución en las prácticas de migración entre Sistemas Operativos: en los días de antaño, cambiar de SO, o mejor dicho, migrar a un nuevo SO, implicaba un esfuerzo considerable ya que requería (por lo menos para un usuario final) trasladar todos los archivos personales a algún tipo de almacenamiento externo y luego de haber actualizado el Sistema Operativo, migrar todos los archivos nuevamente. Hoy en día, el proceso es mucho más simple y la posibilidad de perder algún archivo personal es muy baja, incluso el hardware no debe ser cambiado en la mayoría de los casos.

Esto viene a colación con lo que Intel define en su artículo como el modelo “Sistema Operativo como Servicio”, lo cual, a decir verdad, es un modelo que ha estado hace tiempo desde Windows XP y sus “service packs”, pero que ahora ha cobrado mucha más relevancia debido a los avances tecnológicos actuales (principalmente en lo que es simulación con máquinas virtuales y almacenamiento en la nube). Los sistemas operativos ya no son una pieza de software que cambia a un ritmo muy lento sino que ahora son piezas de software que se encuentran en un proceso de cambio y actualización casi

constante como para mantenerse actualizadas con las tendencias de nuestro mundo, siendo uno de los grandes factores detrás de esta aceleración los avances en seguridad.
Intel menciona en su artículo que ésta es la razón por la cual están cambiando sus políticas y prácticas para el despliegue de nuevos sistemas operativos en la empresa: para adaptarse a este nuevo paradigma en auge.

Estos cambios modifican el circuito que se puede resumir en los siguientes puntos:

1. Análisis de factibilidad: Intel determina si las funcionalidades que provee el nuevo SO demandan que la migración al mismo se haga con mayor rapidez de forma tal de tomar ventaja de las nuevas funcionalidades o si se puede posponer para más tarde.
2. Realizar pruebas con el SO: se prueban las aplicaciones en el nuevo SO y se corrobora que no se produzcan fallos que puedan afectar al trabajo de los empleados.
3. Crear un plan de migración: se crea una imagen del SO que sea estable, se realizan tests sobre ella y se comunica a todas las partes involucradas en el cambio de SO para poder llevar a cabo una migración organizada.
4. Migrar: se comienza a instalar en nuevo SO en las distintas computadoras de la empresa, siguiendo un enfoque “por niveles”. Se comienza por un conjunto reducido de personas llamado “Early Adopters” y se aumenta gradualmente el alcance de la actualización hasta que todos los empleados estén usando el nuevo SO.

Un detalle muy importante a tener en cuenta es el énfasis con que el artículo hace hincapié en el uso de las máquinas virtuales o VMs, tanto para el Testing de aplicaciones sobre el nuevo SO al que se migrará como para el uso de las mismas como Red de Seguridad en caso de que el nuevo SO falle. Es claro que el artículo establece la idea de que “Las VMs son el futuro del Testing”, lo cual es un punto muy importante a tener en cuenta ya que las mismas han encontrado un hábitat perfecto para ellas en el mundo del Testing debido a su naturaleza más bien cerrada: si algo falla durante la ejecución de una VM, la computadora Host no sufriría daños y la VM sólo se detendría forzadamente.

a) Los Aspectos que se relacionan con nuestra Carrera son los aspectos de Testing que se aplican sobre los SO nuevos (uso de Máquinas Virtuales, etc), la importancia de la comunicación durante los procesos laborales y como las herramientas que usamos sufren modificaciones constantemente y en periodos cortos de tiempo.

   Esto afectará mi futura profesión en el hecho que las herramientas que se usen para ese momento no necesariamente vayan a ser las mismas las cuales estoy usando en este momento para mis estudios universitarios

# EJERCICIOS DE PLANIFICACIÓN

**Se asumen los siguientes hechos:**

- El tiempo que importa el Dispatcher en seleccionar un proceso es despreciable a los efectos de la Planificación
- para los ejercicios que hacen uso de Dos Procesadores, se asume la existencia de una Cola de Procesos Listos *compartida*, de la cual los procesadores seleccionan procesos a ejecutar
- para el cálculo de Tiempos se considera al Proceso P2 como dos Procesos por separado, debido a que hace uso de dos Threads a nivel de Kernel, el Tiempo Total para el proceso P2 será la suma de los dos subprocesos “P2a” y “P2b”

Se podrán apreciar los Diagramas de Gantt asociados a estos ejercicios en la Sección Anexo

a) Tiempos:
   i) Espera(P1) = 5
   ii) Retorno(P1) = 12
   iii) Espera(P2a) = 9
   iv) Retorno(P2a) = 9
   v) Espera(P2b) =6
   vi) Retorno(P2b) = 12
   vii) Espera(P3) = 10
   viii) Retorno(P3) = 12
b) Tiempos:
   i) Espera(P1) = 11
   ii) Retorno(P1) = 3
   iii) Espera(P2a) = 11
   iv) Retorno(P2a) = 14
   v) Espera(P2b) = 15
   vi) Retorno(P2b) = 18
   vii) Espera(P3) = 1
   viii) Retorno(P3) = 3

**A DOS PROCESADORES**

a) Tiempos:
   i) Espera(P1) = 1
   ii) Retorno(P1) = 9
   iii) Espera(P2a) = 4
   iv) Retorno(P2a) = 7
   v) Espera(P2b) = 8
   vi) Retorno(P2b) = 11
   vii) Espera(P3) = 9
   viii) Retorno(P3) = 11

b) Tiempos:
   i) Espera(P1) = 2
   ii) Retorno(P1) = 10
   iii) Espera(P2a) = 9
   iv) Retorno(P2a) = 12
   v) Espera(P2b) = 4
   vi) Retorno(P2b) = 7
   vii) Espera(P3) = 5
   viii) Retorno(P3) = 7

De analizar los resultados obtenidos, se percata de lo siguiente:

- Hay una mejora en los tiempos de retorno y espera del Proceso P3 bajo la política SJF Apropiativo con respecto a la Política Round-Robin a costa de aumentar los tiempos de Retorno y Espera de los demás procesos
- La Política Round-Robin es la que mejor distribuye los tiempos, intentado dar un tiempo de ejecución equitativo para todos los procesos, esto se debe a su política de usar un Quanto de tiempo previamente definido
- Al usar dos Procesadores, se notan mejoras en los tiempos de Retorno y Espera para todos los procesos para las dos políticas

# ANEXO

## REFERENCIAS:

E = “El Proceso se encuentra ejecutando en esta Unidad de Tiempo”

L = “El Proceso se encuentra en la Cola de Listos en esta Unidad de Tiempo”

B = “El Proceso se encuentra en la Cola de Bloqueados, esperando Entrada/Salida, en esta Unidad de Tiempo”

X | Y = sean X e Y estados Posibles de un Proceso (E,L o B), se refiere a estos “Estados de Transición” como. “El Proceso estaba en el Estado X y para esta Unidad de Tiempo se encuentra haciendo Y”

FIN Ejec = “El Proceso YA TERMINÓ su ejecución para esta Unidad de Tiempo”

## A UN PROCESADOR

a) Política Round-Robin

| PROCESO | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | E | E | E | E \| L | L | L | L | L | L | L \| E | E | FIN Ejec | | | | | |
| | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| P2a | | | | L | L \| E | E | E \| B | B | B | B | B \| L | L \| E | FIN Ejec | | | | | |
| | | | | | | | | | | | | | | | | | | |
| P2b | | | | L | L | L | L \| E | E | E \| B | B | B | B | B | B | B | B \| E | FIN Ejec | |
| | | | | | | | | | | | | | | | | | | |
| P3 | | | | | L | L | L | L | L \| E | E \| B | B | B | B | B | B | B \| E | FIN Ejec | |
| | | | | | | | | | | | | | | | | | | |
| TIEMPO | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | |

b) Política SJF Apropiativo

| PROCESO | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | E | E | E | E \| L | L | L \| E | E \| L | L \| E | E | E | E | FIN Ejec | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| P2a | | | | E | E \| L | L | L | L | L | L | L | L | L \| E | E \| B | B | B | B | B \| E | FIN Ejec | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| P2b | | | | L | L | L | L | L | L | L | L | L | L | L \| E | E | E \| B | B | B | B | B | B | B \| E | FIN Ejec |
| | | | | | | | | | | | | | | | | | | | | | | | |
| P3 | | | | | E | E \| B | B \| E | FIN Ejec | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| TIEMPO | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | |

## A DOS PROCESADORES

a) Política Round-Robin

| PROCESO | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | E | E | E | E | E \| L | L \| E | E | E | E | FIN Ejec | | | | | | |
| | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | |
| P2a | | | | E | E | E \| B | B | B | B | B \| E | FIN Ejec | | | | | |
| | | | | | | | | | | | | | | | | |
| P2b | | | | L | L \| E | E | E \| B | B | B | B | B | B | B | B | B \| E | FIN Ejec | |
| | | | | | | | | | | | | | | | | |
| P3 | | | | | L | L \| E | E \| B | B | B | B | B | B | B | B | B | B \| E | FIN Ejec |
| | | | | | | | | | | | | | | | | |
| TIEMPO | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

b) Política SJF Apropiativo

| PROCESO | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | E | E | E \| L | L | L \| E | E | E | E | E | FIN Ejec | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| P2a | | | | E \| L | L \| E | E \| B | B | B | B | B | B | B | B | B \| E | FIN Ejec |
| | | | | | | | | | | | | | | | |
| P2b | | | | E | E \| B | B | B | B \| E | E | FIN Ejec | | | | | |
| | | | | | | | | | | | | | | | |
| P3 | | | | E | E \| B | B | B | B | B | B \| E | FIN Ejec | | | | |
| | | | | | | | | | | | | | | | |
| TIEMPO | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |