

[Área personal](#) / [Mis cursos](#) / [Técnicas Digitales III \(Práctica y Exámenes\) 2021](#) / [Parcial 2](#) / [Parcial 2](#)

Comenzado el	jueves, 24 de junio de 2021, 19:13
Estado	Finalizado
Finalizado en	jueves, 24 de junio de 2021, 19:41
Tiempo empleado	27 minutos 29 segundos
Puntos	1,49/10,00
Calificación	1,49 de 10,00 (15%)

Pregunta **1**

Incorrecta

Puntúa 0,00 sobre 0,73

La inversión de prioridad:

- ☐ a. Se puede dar entre dos tareas de distinta prioridad que utilicen una cola de mensajes compartida.
- ☒ b. Se puede dar entre dos tareas de distinta prioridad que utilicen un semáforo binario compartido. ✓
- ☒ c. Intercambia las prioridades de las tareas involucradas, quedando con menos prioridad la tarea que era más prioritaria. ✗
- ☐ d. Siempre aumenta la latencia de la tarea más prioritaria.
- ☐ e. Puede ser evitada utilizando un mutex en lugar de un semáforo binario en FreeRTOS.

Respuesta incorrecta.

Las respuestas correctas son: Se puede dar entre dos tareas de distinta prioridad que utilicen un semáforo binario compartido., Se puede dar entre dos tareas de distinta prioridad que utilicen una cola de mensajes compartida., Siempre aumenta la latencia de la tarea más prioritaria.

Pregunta **2**

Incorrecta

Puntúa 0,00 sobre 0,71

Seleccione lo que corresponda respecto a Administración de memoria con mapas de bits, para particiones variables.

Seleccione una o más de una:

- ☐ a. La fragmentación externa depende del tamaño de la unidad de asignación de memoria.
- ☐ b. Cuanto mayor sea la unidad de asignación de memoria, mayor será el mapa de bits.
- ☐ c. Es rápida la asignación de memoria y lenta la liberación de memoria.
- ☒ d. La asignación y liberación de memoria demoran el mismo tiempo. ✗
- ☐ e. Cuanto menor sea la unidad de asignación, menor será la fragmentación interna.

Respuesta incorrecta.

La respuesta correcta es: Cuanto menor sea la unidad de asignación, menor será la fragmentación interna.

Pregunta 3

Incorrecta

Puntúa 0,00 sobre 0,71

Indique cual de las siguientes afirmaciones son verdaderas cuando se ejecuta el comando `pthread_create()`.

Seleccione una o más de una:

- ☐ a. Si no hubo error, almacena en un buffer indicado por el primer argumento el TID,
- ☐ b. Es posible configurar el tamaño de stack y la prioridad del hilo en el segundo argumento de la llamada.
- ☐ c. Si la creación es exitosa, el hilo ejecuta la función indicada como tercer argumento.
- ☐ d. Si la creación del hilo no es exitosa, retorna el número de error.
- ☒ e. Si hubo error, retorna -1 y no es exitosa la creación del hilo. ✗
- ☒ f. Si no hubo error, retorna el TID (Thread ID) del hilo que crea. ✗

Respuesta incorrecta.

Las respuestas correctas son: Es posible configurar el tamaño de stack y la prioridad del hilo en el segundo argumento de la llamada., Si la creación es exitosa, el hilo ejecuta la función indicada como tercer argumento., Si la creación del hilo no es exitosa, retorna el número de error., Si no hubo error, almacena en un buffer indicado por el primer argumento el TID,

Pregunta 4

Correcta

Puntúa 0,71 sobre 0,71

Seleccione lo verdadero respecto a la técnica de paginación para memoria virtual.

Seleccione una o más de una:

- ☐ a. Se divide la memoria física en páginas de tamaño igual y potencia de 2.
- ☐ b. Al producirse un fallo de página siempre se debe ejecutar un algoritmo de reemplazo de páginas.
- ☒ c. El bus de direcciones del procesador no está directamente conectado a la memoria física. ✓
- ☒ d. Solo crea un espacio de direcciones para cada programa en ejecución. ✓
- ☐ e. No permite ejecutar programas más grandes que la memoria física.

Respuesta correcta

Las respuestas correctas son: Solo crea un espacio de direcciones para cada programa en ejecución., El bus de direcciones del procesador no está directamente conectado a la memoria física.

Pregunta 5

Parcialmente correcta

Puntúa 0,43 sobre 0,72

Seleccione lo correcto referido a la implementación foreground/background (primer plano/segundo plano) para RTS.

Seleccione una o más de una:

- ☐ a. La latencia en estos sistemas está definida por el tiempo que demora en ejecutar el bucle de foreground.
- ☒ b. La latencia en estos sistemas dependerá de lo que demore la ejecución de las rutinas de atención de interrupción. ✓
- ☒ c. Para solucionar posibles problemas de incoherencia de datos se puede hacer uso de semáforos o mutex. ✗
- ☒ d. Pueden tener incoherencia si los datos se comparten entre tareas de primer plano. ✗
- ☐ e. Los sistemas foreground/background tienen mayor eficiencia que los sistemas scan loop.
- ☒ f. Pueden tener incoherencia si los datos se comparten entre rutinas de atención de interrupción y tareas de primer plano. ✓
- ☐ g. La ventaja de estos sistemas respecto de bucle scan es que son mas sencillos de programar.

Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

Las respuestas correctas son: La latencia en estos sistemas dependerá de lo que demore la ejecución de las rutinas de atención de interrupción., Pueden tener incoherencia si los datos se comparten entre rutinas de atención de interrupción y tareas de primer plano.

Pregunta **6**

Parcialmente correcta

Puntúa 0,17 sobre 1,00

Un proceso posee 3 hilos: main, hilo0, hilo1. Los 3 hilos incrementan las variables total y total1.

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- El hilo1 debe esperar al hilo0 antes de acceder a las variables total y total1,
- 3- El hilo main debe esperar al hilo1 antes de acceder a las variables total y total1.
- 4- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;

void *hilo0 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    for(t=0; t < 100000 ; t++){
        total=total+numero;
        total1=total1+1;
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int j, a[5]={0};
    total=10000;
    total1=100000;
    a[0]=10;
    a[1]=2;
    a[3]=1;
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+1;
    }
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total= , Total1=
*/
```

pthread_exit(NULL); 1120000 pthread_join(h[0],NULL); pthread_detach(pthread_self()); 3200000 pthread_yield(); 1210000
pthread_join(h[2],NULL); pthread_join(h[1],NULL); 11200000 pthread_mutexattr_init(&mtxattr); 320000 //línea en blanco

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

La respuesta correcta es:

Un proceso posee 3 hilos: main, hilo0, hilo1. Los 3 hilos incrementan las variables total y total1.

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- El hilo1 debe esperar al hilo0 antes de acceder a las variables total y total1,
- 3- El hilo main debe esperar al hilo1 antes de acceder a las variables total y total1.
- 4- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;

void *hilo0 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    [//linea en blanco]
    for(t=0; t < 100000 ; t++){
        total=total+numero;
        total1=total1+1;
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    [pthread_join(h[0],NULL);]
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int j, a[5]={0};
    total=10000;
    total1=100000;
    a[0]=10;
    a[1]=2;
    a[3]=1;
    [//linea en blanco]
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
    [pthread_join(h[1],NULL);]
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+1;
    }
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total= [1120000] ,Total1= [320000]
*/
```

Pregunta 7

Sin contestar

Puntúa como 1,00

Un proceso posee 1+nh hilos (main + nh). Los hilos incrementan las variables total.

1- Las sumas se sincronizan con un semáforo sin nombre.

2- El hilo main espera a que el resto de los hilos terminen y muestra el valor de total.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>

int t, total, c;
sem_t sem;

void *hilo0 (void * nro) {
    int numero, j;
    numero = *(int*)nro;
    for(j=0; j < c; j++){
        sem_wait(&sem);
        total = total + numero;
    }
    pthread_exit(NULL);
}

int main()    {
    int nh,j,a[1];
    a[0]=10;
    nh = 10;
    pthread_t h[nh];
    total = 10000;
    c = 10000;
    for(t=0; t < nh; t++){
        pthread_create(&h[t], NULL, hilo0, (void *) (&a[t]));
    }
    for(t=0; t < nh; t++){
        pthread_join(h[t], NULL);
    }
    sem_destroy(&sem);
    printf("total = %d\n",total);
    pthread_exit(NULL);
}

/*
En consola vemos
total=
*/
```

```
//linea en blanco sem_init(&sem, 0, 1); sem_unlink(&sem); sem_init(&sem, 0, 0); sem_post(&sem); 1010000 pthread_join(h[t],NULL);
102000 sem_init(&sem, 1, 0); pthread_join(hilo[t],NULL); pthread_create(&h[t], NULL, hilo0, (void *) (&a[0]));
pthread_create(&a[t], NULL, hilo0, (void *) (&a[t])) 10010000 pthread_detach(h[t],NULL);
```

Respuesta incorrecta.

La respuesta correcta es:

Un proceso posee 1+nh hilos (main + nh). Los hilos incrementan las variables total.

1- Las sumas se sincronizan con un semáforo sin nombre.

2- El hilo main espera a que el resto de los hilos terminen y muestra el valor de total.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>

int t, total, c;
sem_t sem;

void *hilo0 (void * nro) {
    int numero, j;
    numero = *(int*)nro;
    [//linea en blanco]
    for(j=0; j < c; j++){
        sem_wait(&sem);
        total = total + numero;
        [sem_post(&sem);]
    }
    pthread_exit(NULL);
}

int main()    {
    int nh,j,a[1];
    a[0]=10;
    nh = 10;
    pthread_t h[nh];
    total = 10000;
    c = 10000;
    [sem_init(&sem, 0, 1);]
    for(t=0; t < nh; t++){
        [pthread_create(&h[t], NULL, hilo0, (void *) (&a[0]));]
    }
    for(t=0; t < nh; t++){
        [pthread_join(h[t],NULL);]
    }
    sem_destroy(&sem);
    printf("total = %d\n",total);
    pthread_exit(NULL);
}

/*
En consola vemos
total= [1010000]
*/
```


Pregunta 8

Incorrecta

Puntúa 0,00 sobre 0,71

Indique lo correcto referido a punto de encuentro del evento A respecto del evento B.

Seleccione una o más de una:

- ☐ a. Para forzar la situación de punto de encuentro (para dos eventos) se pueden usar dos semáforo, ambos inicialmente en 1.
- ☐ b. El evento A no debe ocurrir en el mismo momento que el evento B.
- ☒ c. Para forzar la situación de punto de encuentro, la única alternativa es usar un barrier inicializado en 2. ✖
- ☐ d. Para forzar la situación de punto de encuentro (para dos eventos) se puede usar un semáforo, inicialmente en 0.
- ☐ e. El evento A y el evento B pueden ser concurrentes.
- ☐ f. El evento A debe esperar la ocurrencia del evento B, e inversamente.
- ☐ g. Para forzar la situación de punto de encuentro (para dos eventos) se pueden usar dos semáforos, ambos inicialmente en 0.
- ☐ h. Para forzar la situación de punto de encuentro (para dos eventos) se puede usar un mutex, inicialmente en 0.

Respuesta incorrecta.

Las respuestas correctas son: El evento A debe esperar la ocurrencia del evento B, e inversamente., Para forzar la situación de punto de encuentro (para dos eventos) se pueden usar dos semáforos, ambos inicialmente en 0.

Pregunta 9

Sin contestar

Puntúa como 1,00

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea única con la prioridad más baja posible.
- 2- La tarea debe cambiar el estado del LED rojo, enviar por puerto serie el texto recibido como parámetro en su creación y bloquearse durante 800ms.

```

/*****/
const char *pvTaskParameters = "Tarea 1 en ejecución\r\n";
int main( void )
{
    ( NULL, configMINIMAL_STACK_SIZE, (void*)pvTaskParameters, NULL );
    for(;; );
    return 0;
}
/*****/
void vTask1 ( void *pvParameters )
{
    char *pcTaskName;
    const TickType_t xDelay = ( 800 );
    pcTaskName = ( char * )
    {
        vPrintString( pcTaskName );
        Board_LED_Toggle(4); //LED rojo;
    }
}

tskIDLE_PRIORITY+2 xTaskHandle tskIDLE_PRIORITY+1 xPeriod vTaskStartScheduler(); vTask1 vTaskDelayUntil void vTask2
vTaskDelay xLastWakeTime vPrintString vTaskDelete( NULL ) vTaskPrioritySet xDelay void vTask1 tskIDLE_PRIORITY+3
&xLastWakeTime, xPeriod vTask2 xTaskGetTickCount() xTaskCreate for(;;) pvParameters pdMS_TO_TICKS

```

Respuesta incorrecta.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea única con la prioridad más baja posible.
- 2- La tarea debe cambiar el estado del LED rojo, enviar por puerto serie el texto recibido como parámetro en su creación y bloquearse durante 800ms.

```

/*****/
const char *pvTaskParameters = "Tarea 1 en ejecución\r\n";
int main( void )
{
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, (void*)pvTaskParameters, [tskIDLE_PRIORITY+1],
    NULL );

    [vTaskStartScheduler();]
    for(;; );
    return 0;
}

```

```
/* **** */
void vTask1 ( void *pvParameters )
{
    char *pcTaskName;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 800 );
    pcTaskName = ( char * ) [pvParameters];
    [for( ;; )]
    {
        vPrintString( pcTaskName );
        Board_LED_Toggle(4); //LED rojo;
        [vTaskDelay]( [xDelay] );
    }
}
```

Puntuá como 1,00

- 1- La tarea vTask1 debe enviar a través de una cola de mensajes FIFO el valor entero 123 de manera continua.
- 2- En el caso de que la cola esté llena, vTask1 se debe bloquear por tiempo indefinido.
- 3- La tarea vTask2 debe recibir los datos enviados por vTask1.
- 4- En el caso de que la cola esté vacía, vTask2 se debe bloquear por tiempo indefinido.

```
void vTask2 ( void *pvParameters )
{
    int32_t lReceivedValue;
    const TickType_t xDelay = 1000 );

    for ( ;; ) {
        ( xQueue,
        vPrintString( "dato recibido\r\n" );
    }
}
```

```

&IReceivedValue, xDelay | pdMS_TO_TICKS | xHigherPriorityTaskWoken | &IValueToSend, xDelay | xQueueSendToBackFromISR
xQueueSendToBack | &IValueToSend, portMAX_DELAY | xQueue | portYIELD_FROM_ISR | xQueueSendToFrontFromISR
xQueueReceiveFromISR | uxItemSize | uxQueueLength | xQueueReceive | xQueueSendToFront | xDelay | uxQueueLength, uxItemSize
&IValueToSend | portMAX_DELAY | IReceivedValue | &IReceivedValue, portMAX_DELAY | xQueueCreate | IValueToSend

```

- 1- La tarea vTask1 debe enviar a través de una cola de mensajes FIFO el valor entero 123 de manera continua.
- 2- En el caso de que la cola esté llena, vTask1 se debe bloquear por tiempo indefinido.
- 3- La tarea vTask2 debe recibir los datos enviados por vTask1.
- 4- En el caso de que la cola esté vacía, vTask2 se debe bloquear por tiempo indefinido.

```
QueueHandle_t xQueue;
int main( void )
{
    UBaseType_t uxQueueLength = 10;
    UBaseType_t uxItemSize = sizeof( int32_t );

    xQueue = [xQueueCreate]( [uxQueueLength, uxItemSize] );

    /*Creación de tareas, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    int32_t lValueToSend = 123;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        [xQueueSendToBack]( xQueue, [&lValueToSend, portMAX_DELAY] );
    }
}

void vTask2 ( void *pvParameters )
{
    int32_t lReceivedValue;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        [xQueueReceive]( xQueue, [&lReceivedValue, portMAX_DELAY] );
        vPrintString( "Dato recibido\r\n" );
    }
}
```

Pregunta **11**

Sin contestar

Puntúa como 1,00

ENUNCIADO

Modifique el archivo padre.c para que reciba del proceso hijo 3 caracteres mediante la cola de mensajes declarada como MQ_PATH y los imprima por consola.

El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr. Se recomienda que el padre elimine la cola de mensajes luego de leer el mensaje enviado por el proceso hijo.

El proceso hijo recibe como parámetro la variable mqd. Escribe en la cola de mensaje 3 caracteres y cierra la cola de mensaje. Luego de realizar estas acciones, escribe por consola los siguientes mensajes de confirmación:

Hijo en ejecucion...

Hijo: mensaje enviado.

Hijo: cola de mensajes cerrada.

PASOS A SEGUIR

1) Descargue los siguientes 2 archivos en un mismo directorio:

1.a) Código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/5WnAgmncswdASWQ>

1.b) Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/c3AWSk5SBZN2jzj>

2) Complete el archivo padre.c. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr.

3) Copiar y pegar en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4) El binario generado por el comando anterior se ejecuta en consola con:

```
./padre
```

5) Luego que el proceso padre reciba los 3 caracteres los debe imprimir por consola:

```
XXX
```

RESPUESTA DEL EJERCICIO

Copie el número de 3 cifras XXX en la casilla de abajo. Este número es la respuesta del ejercicio.

Respuesta:



La respuesta correcta es: 366

Pregunta **12**

Parcialmente correcta

Puntúa 0,18 sobre 0,71

Indique lo correcto para el algoritmo de planificación Fair-Share.

Seleccione una o más de una:

- ☒ a. Es un sistema proporcional para los usuarios.
- ☐ b. disminuye el tiempo de retorno
- ☐ c. Usa técnicas de envejecimiento.
- ☐ d. Es un algoritmo expropiativo.
- ☐ e. Es un sistema proporcional para las tareas.
- ☒ f. El planificador tiene prioridades preestablecidas.



Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Es un algoritmo expropiativo., Es un sistema proporcional para los usuarios.

[◀ Recuperatorio Parcial 1](#)

Ir a...

[Área personal](#) / [Mis cursos](#) / [Técnicas Digitales III \(Práctica y Exámenes\) 2021](#) / [Parcial 2](#) / [Parcial 2](#)

Comenzado el	jueves, 24 de junio de 2021, 19:00
Estado	Finalizado
Finalizado en	jueves, 24 de junio de 2021, 19:48
Tiempo empleado	48 minutos 8 segundos
Puntos	3,40/10,00
Calificación	3,40 de 10,00 (34%)

Pregunta **1**

Incorrecta

Puntúa 0,00
sobre 0,73

Una lo que corresponda.

Un RTOS no expropiativo

aumenta la latencia de las tareas de primer plano.

✘

Un sistema
background/foreground
respecto de un scan loop

disminuye la latencia de las tareas de primer plano.

✘

Un sistema scan loop

tiene el peor de los rendimientos.

✘

Un RTOS expropiativo
respecto a un RTOS no
expropiativo

puede tener incoherencia solo si se comparten datos entre ISR y tareas de primer plano.

✘

Respuesta incorrecta.

La respuesta correcta es: Un RTOS no expropiativo → puede tener incoherencia solo si se comparten datos entre ISR y tareas de primer plano., Un sistema background/foreground respecto de un scan loop → disminuye la latencia de las tareas de segundo plano., Un sistema scan loop → tiene el mayor de los rendimientos., Un RTOS expropiativo respecto a un RTOS no expropiativo → disminuye la latencia de las tareas de primer plano.

Pregunta **2**Parcialmente
correctaPuntúa 0,24
sobre 0,71

Seleccione lo verdadero respecto a algoritmos de reemplazo de página para memoria virtual.

Seleccione una o más de una:

- ☒ a. El algoritmo NRU es más fácil de implementar que el algoritmo working set. ✓
- ☐ b. El algoritmo NRU es una mejora del algoritmo clock.
- ☒ c. El algoritmo óptimo no se puede implementar genéricamente y no tiene uso. ✗
- ☒ d. El algoritmo wsclock es una mejora del algoritmo working set. ✓
- ☒ e. El algoritmo LRU es una simplificación del NFU y se ejecuta más rápido. ✗

Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

Las respuestas correctas son: El algoritmo NRU es más fácil de implementar que el algoritmo working set., El algoritmo wsclock es una mejora del algoritmo working set.

Pregunta **3**

Correcta

Puntúa 1,00
sobre 1,00**ENUNCIADO**

Modifique el archivo padre.c para que reciba del proceso hijo 3 caracteres mediante la cola de mensajes declarada como MQ_PATH y los imprima por consola.

El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr. Se recomienda que el padre elimine la cola de mensajes luego de leer el mensaje enviado por el proceso hijo.

El proceso hijo recibe como parámetro la variable mqd. Escribe en la cola de mensaje 3 caracteres y cierra la cola de mensaje. Luego de realizar estas acciones, escribe por consola los siguientes mensajes de confirmación:

Hijo en ejecucion...

Hijo: mensaje enviado.

Hijo: cola de mensajes cerrada.

PASOS A SEGUIR

1) Descargue los siguientes 2 archivos en un mismo directorio:

1.a) Código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/5WnAqmnscswdASWQ>

1.b) Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/EPTxT9N7oPgWBBf>

2) Complete el archivo padre.c. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr.

3) Copiar y pegar en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4) El binario generado por el comando anterior se ejecuta en consola con:

```
./padre
```

5) Luego que el proceso padre reciba los 3 caracteres los debe imprimir por consola:

```
XXX
```

RESPUESTA DEL EJERCICIO

Copie el número de 3 cifras XXX en la casilla de abajo. Este número es la respuesta del ejercicio.

Respuesta: 587



La respuesta correcta es: 587

Pregunta 4

Incorrecta

Puntúa 0,00
sobre 0,72

Una lo que corresponda.

Un sistema scan loop

tiene el peor de los rendimientos.



Un RTOS no expropiativo

tiene el mayor de los rendimientos.

un RTOS expropiativo
respecto a un RTOS no
expropiativo

disminuye la latencia de las tareas de segundo plano.

Un sistema
background/foreground
respecto de un scan loop

permite correr tareas en paralelo.



Respuesta incorrecta.

La respuesta correcta es: Un sistema scan loop → tiene el mayor de los rendimientos., Un RTOS no expropiativo → puede tener incoherencia solo si se comparten datos entre ISR y tareas de primer plano., un RTOS expropiativo respecto a un RTOS no expropiativo → disminuye la latencia de las tareas de primer plano., Un sistema background/foreground respecto de un scan loop → disminuye la latencia de las tareas de segundo plano.

Pregunta **5**

Correcta

Puntúa 0,71
sobre 0,71

Indique cuál de los siguientes es un recurso compartido por los hilos de un mismo proceso.

Seleccione una o más de una:

- ☒ a. Variables globales.
- ☐ b. Contador de programa.
- ☒ c. Señales y manejadores de señales.
- ☐ d. Pila (stack).
- ☒ e. Espacio de direcciones.



Respuesta correcta

El objetivo es que comprenda los recursos usados por los hilos de un proceso.

Las respuestas correctas son: Señales y manejadores de señales., Variables globales., Espacio de direcciones.

Pregunta **6**Parcialmente
correctaPuntúa 0,24
sobre 0,71

Indique lo correcto para el algoritmo de planificación por prioridad con múltiples colas.

Seleccione una o más de una:

- ☐ a. Disminuye la cantidad de cambios de contexto.
- ☐ b. Maximiza el rendimiento.
- ☐ c. Usa técnicas de envejecimiento.
- ☒ d. Cuando los procesos se convierten interactivos, se les asigna la mayor prioridad.
- ☐ e. Se asigna distinta cantidad de quantums a cada nivel de prioridad.



Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Cuando los procesos se convierten interactivos, se les asigna la mayor prioridad., Disminuye la cantidad de cambios de contexto., Se asigna distinta cantidad de quantums a cada nivel de prioridad.

Pregunta **7**

Sin contestar

Puntúa como
1,00

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea única con la prioridad más baja posible.
- 2- La tarea debe cambiar el estado del LED rojo, enviar por puerto serie el texto recibido como parámetro en su creación y bloquearse durante 800ms.

```
/* **** */
const char *pvTaskParameters = "Tarea 1 en ejecución\r\n";
int main( void )
{
    (  , NULL, configMINIMAL_STACK_SIZE,
(void*)pvTaskParameters, , NULL );
    for( ;; );
    return 0;
}
/* **** */
void vTask1 ( void *pvParameters )
{
    char *pcTaskName;
    const TickType_t xDelay = ( 800 );
    pcTaskName = ( char * ) ;
    {
        vPrintString( pcTaskName );
        Board_LED_Toggle(4); //LED rojo;
        (  );
    }
}
```

}

vTask1	&xLastWakeTime, xPeriod	pvParameters	vTaskStartScheduler();
vPrintString	void vTask2	void vTask1	vTaskDelay
tskIDLE_PRIORITY+3	xTaskCreate	xTaskGetTickCount()	xTaskHandle
vTaskDelete(NULL)	vTaskDelayUntil	tskIDLE_PRIORITY+2	pdMS_TO_TICKS
for(;;)	tskIDLE_PRIORITY+1	xDelay	vTaskPrioritySet
vTask2	xLastWakeTime	xPeriod	

Respuesta incorrecta.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea única con la prioridad más baja posible.
- 2- La tarea debe cambiar el estado del LED rojo, enviar por puerto serie el texto recibido como parámetro en su creación y bloquearse durante 800ms.

```
/* **** */
const char *pvTaskParameters = "Tarea 1 en ejecución\r\n";
int main( void )
{
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, (void*)pvTaskParameters, [tskIDLE_PRIORITY+1],
    NULL );

    [vTaskStartScheduler();]
    for( ;; );
}
```



```
        return 0;
    }
    /***** */
    void vTask1 ( void *pvParameters )
    {
        char *pcTaskName;
        const TickType_t xDelay = [pdMS_TO_TICKS]( 800 );
        pcTaskName = ( char * ) [pvParameters];
        [for( ;; )]
        {
            vPrintString( pcTaskName );
            Board_LED_Toggle(4); //LED rojo;
            [vTaskDelay]( [xDelay] );
        }
    }
}
```

Pregunta **8**Parcialmente
correctaPuntúa 0,80
sobre 1,00

Dos procesos relacionados se comunican por medio de una cola de mensajes.

- 1- El proceso padre crea al proceso hijo.
- 2- El proceso padre crea la cola de mensajes.
- 3- El proceso padre escribe un mensaje en la cola de mensajes, espera a que el proceso hijo termine y luego terminar él.
- 4- El proceso hijo lee un mensaje de la cola de mensajes, muestra lo leído y termina.

Completar.

```
#include <mqueue.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>

#define BB "Parcial TD III"
#define SHMM "/TD3"

int rc,t;
char a[1024];
mqd_t m;
struct mq_attr at;
```

```

int main() {
    if (fork() == 0) {
        m = mq_open(SHMM, O_RDWR, 0);
        mq_getattr(m, &at);
        mq_receive(m, a, at.mq_msgsize, 0);
        printf("%s\n", a);
        exit(0);
    }
    at.mq_msgsize = sizeof(a);
    at.mq_maxmsg = 5;
    m = mq_open(SHMM, O_RDWR, 0);
    mq_send(m, BB, strlen(BB)+1, 1);
    wait(NULL);
    exit(0);
}

```



mq_receive(m, BB, strlen(BB), 0);

mq_getattr(mqd, &attr.mq_msgsize);

m = mq_open(SHMM, O_WRONLY, 0777, &attr);

mq_close(mqd);

mq_send(m, a, at.mq_msgsize, 0);

m = mq_open(SHMM, O_RDWR, 0777, &at);

m = mq_open(SHMM, O_WRONLY, 0);

//linea en blanco

m = mq_open(SHMM, O_RDWR | O_CREAT, 0777, &at);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 4.

La respuesta correcta es:

Dos procesos relacionados se comunican por medio de una cola de mensajes.

- 1- El proceso padre crea al proceso hijo.
- 2- El proceso padre crea la cola de mensajes.
- 3- El proceso padre escribe un mensaje en la cola de mensajes, espera a que el proceso hijo termine y luego terminar él.
- 4- El proceso hijo lee un mensaje de la cola de mensajes, muestra lo leído y termina.

Completar.

```
#include <mqueue.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>

#define BB "Parcial TD III"
#define SHMM "/TD3"

int rc,t;
char a[1024];
mqd_t m;
struct mq_attr at;
```

```
int main() {
    if (fork() == 0) {
        m = mq_open(SHMM, O_RDWR, 0);
        [mq_getattr(m, &at);]
        [mq_receive(m, a, at.mq_msgsize, 0);]
        [printf("%s\n", a);]
        exit(0);
    }
    at.mq_msgsize = sizeof(a);
    at.mq_maxmsg = 5;
    [m = mq_open(SHMM, O_RDWR | O_CREAT, 0777, &at);]
    [mq_send(m, BB, strlen(BB)+1, 1);]
    wait(NULL);
    exit(0);
}
```

Pregunta 9

Parcialmente
correctaPuntúa 0,18
sobre 0,71

Seleccione lo correcto respecto de gestión de memoria con particiones fijas.

Seleccione una o más de una:

- ☐ a. Todas las particiones deben ser del mismo tamaño para evitar la fragmentación externa.
- ☐ b. Para asignar particiones se pueden usar cola única o múltiples colas.
- ☒ c. Puede existir fragmentación interna. ✓
- ☐ d. Puede existir fragmentación externa.
- ☐ e. Se utiliza la prioridad para asignar particiones con múltiples colas.
- ☒ f. Es conveniente hacer compactación de memoria, aunque es costosa computacionalmente. ✗

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Puede existir fragmentación interna., Para asignar particiones se pueden usar cola única o múltiples colas.

Pregunta **10**Parcialmente
correctaPuntúa 0,24
sobre 0,71

Indique cuál de las siguientes son condiciones necesarias para que exista deadlock entre dos o más eventos.

Seleccione una o más de una:

- ☐ a. Sin contención ni espera entre los eventos.
- ☐ b. Espera circular entre los eventos.
- ☐ c. Sin expropiación de los eventos.
- ☐ d. Serialización de los eventos.
- ☒ e. Exclusión mutua de los eventos.
- ☐ f. Espera activa de los eventos.



Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Exclusión mutua de los eventos., Espera circular entre los eventos., Sin expropiación de los eventos.

Pregunta **11**

Sin contestar

Puntúa como
1,00

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe enviar a través de una cola de mensajes FIFO el valor entero 123 de manera continua.
- 2- En el caso de que la cola esté llena, vTask1 se debe bloquear por tiempo indefinido.
- 3- La tarea vTask2 debe recibir los datos enviados por vTask1.
- 4- En el caso de que la cola esté vacía, vTask2 se debe bloquear por tiempo indefinido.

```
QueueHandle_t xQueue;
int main( void )
{
    UBaseType_t uxQueueLength = 10;
    UBaseType_t uxItemSize = sizeof( int32_t );

    xQueue =  (  );

    /*Creación de tareas, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    int32_t lValueToSend = 123;
    const TickType_t xDelay =  ( 1000 );

    for ( ;; ) {
         ( xQueue,  );
    }
}

void vTask2 ( void *pvParameters )
{
    int32_t lReceivedValue;
```



```

const TickType_t xDelay =  ( 1000 );

for ( ;; ) {
     ( xQueue,  );
    vPrintString( "Dato recibido\r\n" );
}

```

xQueueSendToBack	xQueue	IValueToSend
xQueueReceiveFromISR	xQueueCreate	xQueueSendToBackFromISR
portMAX_DELAY	IReceivedValue	&IValueToSend, portMAX_DELAY
&IValueToSend, xDelay	pdMS_TO_TICKS	xQueueSendToFrontFromISR
uxItemSize	uxQueueLength, uxItemSize	xDelay
&IReceivedValue, xDelay	&IReceivedValue, portMAX_DELAY	portYIELD_FROM_ISR
uxQueueLength	xHigherPriorityTaskWoken	xQueueSendToFront
&IValueToSend	xQueueReceive	

Respuesta incorrecta.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe enviar a través de una cola de mensajes FIFO el valor entero 123 de manera continua.
- 2- En el caso de que la cola esté llena, vTask1 se debe bloquear por tiempo indefinido.
- 3- La tarea vTask2 debe recibir los datos enviados por vTask1.

4- En el caso de que la cola esté vacía, vTask2 se debe bloquear por tiempo indefinido.

```
QueueHandle_t xQueue;

int main( void )
{
    UBaseType_t uxQueueLength = 10;
    UBaseType_t uxItemSize = sizeof( int32_t );

    xQueue = [xQueueCreate]( [uxQueueLength, uxItemSize] );

    /*Creación de tareas, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    int32_t lValueToSend = 123;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        [xQueueSendToBack]( xQueue, [&lValueToSend, portMAX_DELAY] );
    }
}

void vTask2 ( void *pvParameters )
{
    int32_t lReceivedValue;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        [xQueueReceive]( xQueue, [&lReceivedValue, portMAX_DELAY] );
        vPrintString( "Dato recibido\r\n" );
    }
}
```

Pregunta **12**

Sin contestar

Puntúa como
1,00

Un proceso posee $1+nh$ hilos (main + nh). Los hilos incrementan las variables total.

1- Las sumas se sincronizan con un semáforo sin nombre.

2- El hilo main espera a que el resto de los hilos terminen y muestra el valor de total.


Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>
```


```
int t, total, c;
sem_t sem;
```

```
void *hilo0 (void * nro) {
```

```
    int numero, j;
    numero = *(int*)nro;
```

```
    
```

```
    for(j=0; j < c; j++){
        sem_wait(&sem);
        total = total + numero;
```

```
        
```

```
    }
    pthread_exit(NULL);
```

```
}
```

```
int main()    {
```

```
    int nh,j,a[1];
```

```
    a[0]=10;
```

```
    nh = 10;
```

```
    pthread_t h[nh];
```

```
    total = 10000;
```

```
    c = 10000;
```

```
    
```

```
    for(t=0; t < nh; t++){
```

```
        
```

```
    }
```

```
    for(t=0; t < nh; t++){
```

```
        
```

```
}  
sem_destroy(&sem);  
printf("total = %d\n", total);  
pthread_exit(NULL);  
}  
  
/*  
En consola vemos  
total=   
*/
```

//línea en blanco	pthread_create(&h[t], NULL, hilo0, (void *) (&a[0]));
102000	sem_init(&sem, 0, 1);
pthread_join(hilo[t],NULL);	pthread_create(&a[t], NULL, hilo0, (void *) (&a[t]))
sem_init(&sem, 1, 0);	10010000
sem_post(&sem);	sem_unlink(&sem);
pthread_detach(h[t],NULL);	sem_init(&sem, 0, 0);
1010000	pthread_join(h[t],NULL);

Respuesta incorrecta.

La respuesta correcta es:

Un proceso posee 1+nh hilos (main + nh). Los hilos incrementan las variables total.

1- Las sumas se sincronizan con un semáforo sin nombre.

2- El hilo main espera a que el resto de los hilos terminen y muestra el valor de total.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>

int t, total, c;
sem_t sem;

void *hilo0 (void * nro) {
    int numero, j;
    numero = *(int*)nro;
    [//linea en blanco]
    for(j=0; j < c; j++){
        sem_wait(&sem);
        total = total + numero;
        [sem_post(&sem);]
    }
    pthread_exit(NULL);
}

int main()    {
    int nh,j,a[1];
    a[0]=10;
    nh = 10;
    pthread_t h[nh];
    total = 10000;
    c = 10000;
    [sem_init(&sem, 0, 1);]
    for(t=0; t < nh; t++){
        [pthread_create(&h[t], NULL, hilo0, (void *) (&a[0]));]
    }
    for(t=0; t < nh; t++){
        [pthread_join(h[t],NULL);]
    }
    sem_destroy(&sem);
    printf("total = %d\n",total);
}
```

```
pthread_exit(NULL);  
}  
  
/*  
En consola vemos  
total= [1010000]  
*/
```

[◀ Recuperatorio Parcial 1](#)[Ir a...](#)

Comenzado el	jueves, 24 de junio de 2021, 19:00
Estado	Finalizado
Finalizado en	jueves, 24 de junio de 2021, 19:57
Tiempo empleado	56 minutos 54 segundos
Vencido	1 minutos 54 segundos
Puntos	5,32/10,00
Calificación	5,32 de 10,00 (53%)

Pregunta **1**

Parcialmente correcta

Puntúa 0,18 sobre 0,71

Seleccione lo correcto respecto de gestión de memoria con particiones fijas.

Seleccione una o más de una:

- ☐ a. Se utiliza la prioridad para asignar particiones con múltiples colas.
- ☐ b. Es conveniente hacer compactación de memoria, aunque es costosa computacionalmente.
- ☒ c. Todas las particiones deben ser del mismo tamaño para evitar la fragmentación externa.
- ☐ d. Puede existir fragmentación interna.
- ☒ e. Para asignar particiones se pueden usar cola única o múltiples colas.
- ☐ f. Puede existir fragmentación externa.



Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Puede existir fragmentación interna., Para asignar particiones se pueden usar cola única o múltiples colas.

Pregunta **2**

Parcialmente correcta

Puntúa 0,71 sobre 1,00

Un proceso posee 3 hilos (main, hilo0, hilo1).

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- Los hilos hilo1 e hilo2 incrementan las variables total y total1.
- 3- El hilo main incrementa la variable total.
- 4- El hilo main espera a que los hilos terminen y muestra el resultado de total y total1.

Completar:

```

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;
pthread_mutex_t mtx0=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mtx1=PTHREAD_MUTEX_INITIALIZER;

void * hilo1() {
    int t;
     ✓
     ✓
    for(t=0; t < 100000 ; t++) 
        total=total+2;
        total1=total1+1;
    }
    pthread_mutex_unlock(&mtx1);
    pthread_mutex_unlock(&mtx0);
    pthread_exit(NULL);
}

void * hilo2() {
    int t;
    pthread_mutex_lock(&mtx0);
    pthread_mutex_lock(&mtx1);
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+2;
    }
    pthread_mutex_unlock(&mtx1);
    pthread_mutex_unlock(&mtx0);
    pthread_exit(NULL);
}

int main() {
    int t;
    total=20000;
    total1=1000;
    pthread_create (&h[0], NULL, hilo1, NULL);
    pthread_create (&h[1], NULL, hilo2, NULL);
     ✓
    for(t=0; t < 1000000 ; t++){
         ✓
        total=total+1; 
        total=total+1;
    }
    pthread_mutex_unlock(&mtx0);
    pthread_join(h[0],NULL);
     ✗
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total=  ✗ , Total1=  ✓
*/

```

2230000	//linea en blanco	pthread_mutex_unlock(&mtx0);	22300000
2130000		pthread_mutex_lock(&mtx1);	1210000
pthread_mutex_unlock(&mtx1);	2300000		pthread_mutex_lock(&mtx0);
2200000	121000	pthread_join(h[0],NULL);	pthread_join(h[1],NULL);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 5.

La respuesta correcta es:

Un proceso posee 3 hilos (main, hilo0, hilo1).

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- Los hilos hilo1 e hilo2 incrementan las variables total y total1.
- 3- El hilo main incrementa la variable total.
- 4- El hilo main espera a que los hilos terminen y muestra el resultado de total y total1.

Completar:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;
pthread_mutex_t mtx0=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mtx1=PTHREAD_MUTEX_INITIALIZER;

void * hilo1() {
    int t;
    [pthread_mutex_lock(&mtx0);]
    [pthread_mutex_lock(&mtx1);]
    for(t=0; t < 100000 ; t++){
        total=total+2;
        total1=total1+1;
    }
    pthread_mutex_unlock(&mtx1);
    pthread_mutex_unlock(&mtx0);
    pthread_exit(NULL);
}

void * hilo2() {
    int t;
    pthread_mutex_lock(&mtx0);
    pthread_mutex_lock(&mtx1);
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+2;
    }
    pthread_mutex_unlock(&mtx1);
    pthread_mutex_unlock(&mtx0);
    pthread_exit(NULL);
}

int main() {
    int t;
    total=20000;
    total1=1000;
    pthread_create (&h[0], NULL, hilo1, NULL);
    pthread_create (&h[1], NULL, hilo2, NULL);
    [pthread_mutex_lock(&mtx0);]
    for(t=0; t < 1000000 ; t++){
        [//linea en blanco]
        total=total+1;
        total=total+1;
    }
    pthread_mutex_unlock(&mtx0);
    pthread_join(h[0],NULL);
    [pthread_join(h[1],NULL);]
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total= [2230000] ,Total1= [121000]
*/
```

Pregunta **3**

Correcta

Puntúa 0,73 sobre 0,73

Seleccione lo correcto respecto a ISR.

- ☐ a. Si la ISR invoca una función del RTOS que pone una tarea prioritaria activa, se debe conmutar a esa tarea.
- ☐ b. La manera de evitar conmutación de tareas dentro de las ISR es deshabilitar las interrupciones.
- ☐ c. La manera de evitar conmutación de tareas dentro de las ISR es agregar un argumento a las llamadas a sistema del RTOS con un timeout.
- ☒ d. Las ISR no deben invocar a funciones del RTOS que conmuten de tareas. ✓
- ☐ e. Las ISR no pueden invocar a funciones del RTOS.

Respuesta correcta

La respuesta correcta es: Las ISR no deben invocar a funciones del RTOS que conmuten de tareas.

Pregunta **4**

Parcialmente correcta

Puntúa 0,12 sobre 0,71

De las siguientes afirmaciones, seleccione la o las que crea que son verdaderas respecto al uso de hilos.

Seleccione una o más de una:

- ☐ a. Permiten el paralelismo real en computadoras de más de un núcleo
- ☒ b. La creación y destrucción de hilos es más rápida que la de subrutinas. ✗
- ☒ c. Ante un error en tiempo de ejecución en el código del hilo, se termina el proceso al que pertenece el hilo. ✓
- ☐ d. Todas las afirmaciones solo son válidas para sistemas multiprocesador o multinúcleo.
- ☐ e. Ante un error en tiempo de ejecución en el código del hilo, solo éste termina y su proceso asociado se sigue ejecutando.

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Ante un error en tiempo de ejecución en el código del hilo, se termina el proceso al que pertenece el hilo., Permiten el paralelismo real en computadoras de más de un núcleo

Pregunta **5**

Parcialmente correcta

Puntúa 0,12 sobre 0,71

Seleccione lo verdadero respecto a algoritmos de reemplazo de página para memoria virtual.

Seleccione una o más de una:

- ☒ a. El algoritmo Working Set se implementa con una lista circular y los bits M y R.
- ☒ b. El algoritmo FIFO se implementa con una lista enlazada y los bits M y R.
- ☐ c. El algoritmo NRU se implementa con una lista circular y un contador.
- ☐ d. El algoritmo LRU se implementa con los bits M y R.
- ☐ e. El algoritmo aging es una alternativa de LRU, sin utilizar hardware especial.



Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: El algoritmo Working Set se implementa con una lista circular y los bits M y R., El algoritmo aging es una alternativa de LRU, sin utilizar hardware especial.

Pregunta **6**

Parcialmente correcta

Puntúa 0,36 sobre 0,72

Indique lo correcto respecto a RTS (Real Time Systems).

Seleccione una o más de una:

- ☐ a. Un RTS es aquel en el que la corrección del resultado depende del instante en que se produce o de su validez lógica.
- ☐ b. El uso de un RTOS permite asegurar el determinismo para un RTS.
- ☐ c. El periodo de muestreo (T_s) se refiere al periodo necesario para la adquisición de datos.
- ☐ d. Los RTS no tienen que ejecutarse rápidamente.
- ☒ e. El determinismo en RTS es poder cumplir predeciblemente requisitos temporales.



Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: El determinismo en RTS es poder cumplir predeciblemente requisitos temporales., Los RTS no tienen que ejecutarse rápidamente.

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea vTask1 que se ejecute de manera continua e informe este estado por el puerto serie.
- 2- vTask1 se debe crear en primer lugar.
- 3- Posea una tarea vTask2 que pase a estado "Lista" exactamente cada 750ms y cambie el estado del LED verde.

```
int main( void )
{
    [ ] ✓ ( [ ] ✓ , NULL, configMINIMAL_STACK_SIZE, NULL,
[ ] ✓ , NULL );
    [ ] ✓ ( [ ] ✓ , NULL, configMINIMAL_STACK_SIZE, NULL,
[ ] ✓ , NULL );
    [ ] ✓
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for( ;; ) {
        vPrintString( "Tarea continua en ejecución\r\n" );
    }
}

void vTask2 ( void *pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = [ ] ✓ ( 750 );

    xLastWakeTime = [ ] ✗ ;

    for( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [ ] ✗ ( [ ] ✗ );
    }
}
```

xLastWakeTime	&xLastWakeTime, xPeriod	vTaskDelay	vTaskStartScheduler();
xTaskGetTickCount()	pvParameters		tskIDLE_PRIORITY+2
xTaskHandle	xDelay	void vTask1	void vTask2
vTaskPrioritySet	tskIDLE_PRIORITY+3	vTaskDelete(NULL)	for(;;)
pdMS_TO_TICKS	750		xTaskCreate
	vTask1	vTask2	xPeriod
vPrintString	vTaskDelayUntil		

Respuesta parcialmente correcta.

Ha seleccionado correctamente 8.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea vTask1 que se ejecute de manera continua e informe este estado por el puerto serie.
- 2- vTask1 se debe crear en primer lugar.
- 3- Posea una tarea vTask2 que pase a estado "Lista" exactamente cada 750ms y cambie el estado del LED verde.

```
int main( void )
{
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+2], NULL );
    [xTaskCreate]( [vTask2], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+3], NULL );

    [vTaskStartScheduler();]

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for( ;; ) {
        vPrintString( "Tarea continua en ejecución\r\n" );;;
    }
}

void vTask2 ( void *pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = [pdMS_TO_TICKS]( 750 );

    xLastWakeTime = [xTaskGetTickCount()];
    for( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [vTaskDelayUntil]( [&xLastWakeTime, xPeriod] );
    }
}
```


Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Ambas tareas compiten por el uso del puerto serie, un recurso crítico.
- 2- vTask1 espera indefinidamente la liberación del recurso.
- 3- vTask2 espera 1 segundo, si no se libera el recurso cambia de estado el LED rojo.

```
SemaphoreHandle_t xMutex;  
SemaphoreHandle_t xBinarySemaphore;  
int main( void )  
{  
    [ ] ✓ = [ ] ✓ ;  
  
    /*Creación de tareas, habilitación de interrupciones, inicio planificador*/  
  
    for( ;; );  
    return 0;  
}  
  
void vTask1 ( void *pvParameters )  
{  
    const TickType_t xDelay = [ ] ✓ ( 1000 );  
    for ( ;; ) {  
        if( [ ] ✓ ( [ ] ✓ ) == pdTRUE) {  
            vPrintString( "Acceso critico vTask1\r\n" );  
            [ ] ✓ ( [ ] ✓ );  
        } else Board_LED_Toggle(5); //LED verde  
    }  
}  
  
void vTask2 ( void *pvParameters )  
{  
    const TickType_t xDelay = [ ] ✓ ( 1000 );  
    for ( ;; ) {  
        if( [ ] ✓ ( [ ] ✓ ) == pdTRUE) {  
            vPrintString( "Acceso critico vTask2\r\n" );  
            [ ] ✓ ( [ ] ✓ );  
        } else Board_LED_Toggle(4); //LED rojo  
    }  
}
```

	pdMS_TO_TICKS	xBinarySemaphore, portMAX_DELAY
xBinarySemaphore	xDelay	xMutex, xDelay
	xSemaphoreTake	xBinarySemaphore, xDelay
xHigherPriorityTaskWoken		xSemaphoreCreateMutex()
	xMutex, portMAX_DELAY	xSemaphoreCreateBinary()
	xMutex	
xSemaphoreGive	xSemaphoreGiveFromISR	portYIELD_FROM_ISR
xSemaphoreTakeFromISR		

Respuesta correcta

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Ambas tareas compiten por el uso del puerto serie, un recurso crítico.
- 2- vTask1 espera indefinidamente la liberación del recurso.
- 3- vTask2 espera 1 segundo, si no se libera el recurso cambia de estado el LED rojo.

```
SemaphoreHandle_t xMutex;
SemaphoreHandle_t xBinarySemaphore;
int main( void )
{
    [xMutex] = [xSemaphoreCreateMutex()];

    /*Creación de tareas, habilitación de interrupciones, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );
    for ( ;; ) {
        if( [xSemaphoreTake]( [xMutex, portMAX_DELAY] ) == pdTRUE) {
            vPrintString( "Acceso crítico vTask1\r\n" );
            [xSemaphoreGive]( [xMutex] );
        } else Board_LED_Toggle(5); //LED verde
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        if( [xSemaphoreTake]( [xMutex, xDelay] ) == pdTRUE) {
            vPrintString( "Acceso critico vTask2\r\n" );
            [xSemaphoreGive]( [xMutex] );
        } else Board_LED_Toggle(4); //LED rojo
    }
}
```

ENUNCIADO

Modifique el archivo padre.c para que reciba del proceso hijo 3 caracteres mediante la cola de mensajes declarada como MQ_PATH y los imprima por consola.

El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr. Se recomienda que el padre elimine la cola de mensajes luego de leer el mensaje enviado por el proceso hijo.

El proceso hijo recibe como parámetro la variable mqd. Escribe en la cola de mensaje 3 caracteres y cierra la cola de mensaje. Luego de realizar estas acciones, escribe por consola los siguientes mensajes de confirmación:

Hijo en ejecucion...

Hijo: mensaje enviado.

Hijo: cola de mensajes cerrada.

PASOS A SEGUIR

1) Descargue los siguientes 2 archivos en un mismo directorio:

1.a) Código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/5WnAqmnscswdASWQ>

1.b) Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/EPTxT9N7oPqWBBf>

2) Complete el archivo padre.c. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr.

3) Copiar y pegar en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4) El binario generado por el comando anterior se ejecuta en consola con:

```
./padre
```

5) Luego que el proceso padre reciba los 3 caracteres los debe imprimir por consola:

```
XXX
```

RESPUESTA DEL EJERCICIO

Copie el número de 3 cifras XXX en la casilla de abajo. Este número es la respuesta del ejercicio.

Respuesta:



La respuesta correcta es: 587



Pregunta **10**

Correcta

Puntúa 0,71 sobre 0,71

Indique lo correcto referido a concurrencia del evento A respecto del evento B.

Seleccione una o más de una:

- ☐ a. Para forzar la situación de concurrencia (para dos eventos) se puede usar un mutex, inicialmente en 0.
- ☒ b. No es necesario forzar la situación de concurrencia, ni usar mutex, barrier o semáforos. 
- ☐ c. El evento A debe esperar la ocurrencia del evento B, e inversamente.
- ☒ d. El evento A y el evento B pueden ejecutarse simultáneamente en caso de usar sistemas multitarea. 
- ☐ e. Para forzar la situación de concurrencia (para dos eventos) se puede usar un mutex para cada uno, ambos inicialmente en 1.
- ☐ f. Para forzar la situación de concurrencia (para dos eventos) se puede usar barrier, inicialmente en 2.
- ☐ g. El evento A no debe ocurrir en el mismo momento que el evento B.
- ☐ h. Para forzar la situación de concurrencia (para dos eventos) se puede usar un semáforo, inicialmente en 1.

Respuesta correcta

Las respuestas correctas son: El evento A y el evento B pueden ejecutarse simultáneamente en caso de usar sistemas multitarea., No es necesario forzar la situación de concurrencia, ni usar mutex, barrier o semáforos.

Pregunta **11**

Parcialmente correcta

Puntúa 0,67 sobre 1,00

Un proceso posee 3 hilos: main, hilo0, hilo1. Los 3 hilos incrementan las variables total y total1.

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- El hilo1 debe esperar al hilo0 antes de acceder a las variables total y total1,
- 3- El hilo main debe esperar al hilo1 antes de acceder a las variables total y total1.
- 4- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;

void *hilo0 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
     ✓
    for(t=0; t < 100000 ; t++){
        total=total+numero;
        total1=total1+1;
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
     ✓
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int j, a[5]={0};
    total=10000;
    total1=100000;
    a[0]=10;
    a[1]=2;
    a[3]=1;
     ✓
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
     ✓
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+1;
    }
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total=  ✗ ,Total1=  ✗
*/
```

1120000	pthread_exit(NULL);	11200000
3200000	320000	pthread_join(h[1],NULL);
	pthread_join(h[0],NULL);	pthread_mutexattr_init(&mtxattr);
pthread_yield();	1210000	pthread_detach(pthread_self());
	//linea en blanco	pthread_join(h[2],NULL);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 4.

La respuesta correcta es:

Un proceso posee 3 hilos: main, hilo0, hilo1. Los 3 hilos incrementan las variables total y total1.

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- El hilo1 debe esperar al hilo0 antes de acceder a las variables total y total1,
- 3- El hilo main debe esperar al hilo1 antes de acceder a las variables total y total1.
- 4- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

```

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;

void *hilo0 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    [//linea en blanco]
    for(t=0; t < 100000 ; t++){
        total=total+numero;
        total1=total1+1;
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    [pthread_join(h[0],NULL);]
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int j, a[5]={0};
    total=10000;
    total1=100000;
    a[0]=10;
    a[1]=2;
    a[3]=1;
    [//linea en blanco]
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
    [pthread_join(h[1],NULL);]
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+1;
    }
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total= [1120000] ,Total1= [320000]
*/

```

Pregunta **12**

Sin contestar

Puntúa como 0,71

Indique lo correcto para el algoritmo de planificación Fair-Share.

Seleccione una o más de una:

- ☐ a. Es un sistema proporcional para las tareas.
- ☐ b. Usa técnicas de envejecimiento.
- ☐ c. disminuye el tiempo de retorno
- ☐ d. Es un sistema proporcional para los usuarios.
- ☐ e. Es un algoritmo expropiativo.
- ☐ f. El planificador tiene prioridades preestablecidas.

Respuesta incorrecta.

Las respuestas correctas son: Es un algoritmo expropiativo., Es un sistema proporcional para los usuarios.

[◀ Recuperatorio Parcial 1](#)

Ir a...

Comenzado el	jueves, 24 de junio de 2021, 19:00
Estado	Finalizado
Finalizado en	jueves, 24 de junio de 2021, 19:55
Tiempo empleado	54 minutos 58 segundos
Puntos	5,27/10,00
Calificación	5,27 de 10,00 (53%)

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe enviar a través de una cola de mensajes FIFO el valor entero 123 de manera continua.
- 2- En el caso de que la cola esté llena, vTask1 se debe bloquear durante 1 segundo.
- 3- La tarea vTask2 debe recibir los datos enviados por vTask1.
- 4- En el caso de que la cola esté vacía, vTask2 se debe bloquear durante 500ms.

```
QueueHandle_t xQueue;
int main( void )
{
    UBaseType_t uxQueueLength = 10;
    UBaseType_t uxItemSize = sizeof( int32_t );

    xQueue = [input] ✓ ( [input] ✓ );

    /*Creación de tareas, inicio planificador*/
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    int32_t lValueToSend = 123;
    const TickType_t xDelay = [input] ✓ ( 1000 );

    for ( ;; ) {
        [input] ✗ ( xQueue, [input] ✓ );
    }
}

void vTask2 ( void *pvParameters )
{
    int32_t lReceivedValue;
    const TickType_t xDelay = [input] ✓ ( 500 );

    for ( ;; ) {
        [input] ✓ ( xQueue, [input] ✓ );
        vPrintString( "Dato recibido\r\n" );
    }
}
```

xQueueReceive	lValueToSend	&lValueToSend, portMAX_DELAY
&lReceivedValue, portMAX_DELAY	uxItemSize	xHigherPriorityTaskWoken
&lValueToSend	uxQueueLength	xQueue
xQueueReceiveFromISR		uxQueueLength, uxItemSize
xQueueSendToBackFromISR	xQueueSendToBack	xDelay
portMAX_DELAY		xQueueCreate
	xQueueSendToFront	&lReceivedValue, xDelay
	&lValueToSend, xDelay	

pdMS_TO_TICKS	lReceivedValue	xQueueSendToFrontFromISR
portYIELD_FROM_ISR		

Respuesta parcialmente correcta.

Ha seleccionado correctamente 7.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe enviar a través de una cola de mensajes FIFO el valor entero 123 de manera continua.
- 2- En el caso de que la cola esté llena, vTask1 se debe bloquear durante 1 segundo.
- 3- La tarea vTask2 debe recibir los datos enviados por vTask1.
- 4- En el caso de que la cola esté vacía, vTask2 se debe bloquear durante 500ms.

```
QueueHandle_t xQueue;
int main( void )
{
    UBaseType_t uxQueueLength = 10;
    UBaseType_t uxItemSize = sizeof( int32_t );

    xQueue = [xQueueCreate]( [uxQueueLength, uxItemSize] );

    /*Creación de tareas, inicio planificador*/
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    int32_t lValueToSend = 123;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        [xQueueSendToBack]( xQueue, [&lValueToSend, xDelay] );
    }
}

void vTask2 ( void *pvParameters )
{
    int32_t lReceivedValue;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 500 );

    for ( ;; ) {
        [xQueueReceive]( xQueue, [&lReceivedValue, portMAX_DELAY] );
        vPrintString( "Dato recibido\r\n" );
    }
}
```

Pregunta **2**

Incorrecta

Puntúa 0,00 sobre 0,71

De las siguientes afirmaciones, seleccione la o las que crea que son verdaderas respecto al uso de hilos.

Seleccione una o más de una:

- ☒ a. Permiten el paralelismo real en computadoras de mas de un núcleo ✓
- ☐ b. Ante un error en tiempo de ejecución en el código del hilo, se termina el proceso al que perternece el hilo.
- ☒ c. La creación y destrucción de hilos es más rápida que la de subrutinas. ✗
- ☒ d. Ante un error en tiempo de ejecución en el código del hilo, solo éste termina y su proceso asociado se sigue ejecutando. ✗
- ☐ e. Todas las afirmaciones solo son válidas para sistemas multiprocesador o multinúcleo.

Respuesta incorrecta.

Las respuestas correctas son: Ante un error en tiempo de ejecución en el código del hilo, se termina el proceso al que perternece el hilo., Permiten el paralelismo real en computadoras de mas de un núcleo

Pregunta **3**

Parcialmente correcta

Puntúa 0,54 sobre 0,72

Seleccione lo correcto referido a la implementación foreground/background (primer plano/segundo plano) para RTS.

Seleccione una o más de una:

- ☐ a. La latencia en estos sistemas está definida por el tiempo que demora en ejecutar el bucle de foreground.
- ☐ b. Los sistemas foreground/background tienen mayor eficiencia que los sistemas scan loop.
- ☒ c. La ventaja de estos sistemas respecto de bucle scan es que son mas sencillos de programar. ✗
- ☐ d. Para solucionar posibles problemas de incoherencia de datos se puede hacer uso de semáforos o mutex.
- ☒ e. La latencia en estos sistemas está definida por las rutinas de atención de interrupción. ✓

Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

La respuesta correcta es: La latencia en estos sistemas está definida por las rutinas de atención de interrupción.

Pregunta **4**

Incorrecta

Puntúa 0,00 sobre 0,71

Indique lo correcto para el algoritmo de planificación Fair-Share.

Seleccione una o más de una:

- ☒ a. Es un sistema proporcional para las tareas.
- ☒ b. El planificador tiene prioridades preestablecidas.
- ☐ c. Usa técnicas de envejecimiento.
- ☐ d. Es un algoritmo expropiativo.
- ☐ e. disminuye el tiempo de retorno
- ☐ f. Es un sistema proporcional para los usuarios.

✖

✖

Respuesta incorrecta.

Las respuestas correctas son: Es un algoritmo expropiativo., Es un sistema proporcional para los usuarios.

Pregunta **5**

Incorrecta

Puntúa 0,00 sobre 1,00

ENUNCIADO

Modifique el archivo padre.c para que reciba del proceso hijo 3 caracteres mediante la cola de mensajes declarada como MQ_PATH y los imprima por consola.

El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr. Se recomienda que el padre elimine la cola de mensajes luego de leer el mensaje enviado por el proceso hijo.

El proceso hijo recibe como parámetro la variable mqd. Escribe en la cola de mensaje 3 caracteres y cierra la cola de mensaje. Luego de realizar estas acciones, escribe por consola los siguientes mensajes de confirmación:

Hijo en ejecucion...

Hijo: mensaje enviado.

Hijo: cola de mensajes cerrada.

PASOS A SEGUIR

1) Descargue los siguientes 2 archivos en un mismo directorio:

1.a) Código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/5WnAqmnswdASWQ>

1.b) Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/mcLnwrPaQg7PFNf>

2) Complete el archivo padre.c. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr.

3) Copiar y pegar en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4) El binario generado por el comando anterior se ejecuta en consola con:

```
./padre
```

5) Luego que el proceso padre reciba los 3 caracteres los debe imprimir por consola:

```
XXX
```

RESPUESTA DEL EJERCICIO

Copie el número de 3 cifras XXX en la casilla de abajo. Este número es la respuesta del ejercicio.

Respuesta:



La respuesta correcta es: 470

Pregunta **6**

Correcta

Puntúa 0,71 sobre 0,71

Indique lo correcto referido a punto de encuentro del evento A respecto del evento B.

Seleccione una o más de una:

- ☒ a. El evento A debe esperar la ocurrencia del evento B, e inversamente. ✓
- ☐ b. El evento A y el evento B pueden ser concurrentes.
- ☐ c. Para forzar la situación de punto de encuentro, la única alternativa es usar un barrier inicializado en 2.
- ☐ d. El evento A no debe ocurrir en el mismo momento que el evento B.
- ☐ e. Para forzar la situación de punto de encuentro (para dos eventos) se pueden usar dos semáforo, ambos inicialmente en 1.
- ☒ f. Para forzar la situación de punto de encuentro (para dos eventos) se pueden usar dos semáforos, ambos inicialmente en 0. ✓
- ☐ g. Para forzar la situación de punto de encuentro (para dos eventos) se puede usar un mutex, inicialmente en 0.
- ☐ h. Para forzar la situación de punto de encuentro (para dos eventos) se puede usar un semáforo, inicialmente en 0.

Respuesta correcta

Las respuestas correctas son: El evento A debe esperar la ocurrencia del evento B, e inversamente., Para forzar la situación de punto de encuentro (para dos eventos) se pueden usar dos semáforos, ambos inicialmente en 0.

Pregunta **7**

Correcta

Puntúa 0,71 sobre 0,71

Seleccione lo correcto respecto de gestión de memoria con particiones fijas.

Seleccione una o más de una:

- ☐ a. Se utiliza la prioridad para asignar particiones con múltiples colas.
- ☐ b. Todas las particiones deben ser del mismo tamaño para evitar la fragmentación externa.
- ☒ c. Puede existir fragmentación interna. ✓
- ☐ d. Es conveniente hacer compactación de memoria, aunque es costosa computacionalmente.
- ☒ e. Para asignar particiones se pueden usar cola única o múltiples colas. ✓
- ☐ f. Puede existir fragmentación externa.

Respuesta correcta

Las respuestas correctas son: Puede existir fragmentación interna., Para asignar particiones se pueden usar cola única o múltiples colas.

Pregunta **8**

Parcialmente correcta

Puntúa 0,12 sobre 0,71

Seleccione lo verdadero respecto a algoritmos de reemplazo de página para memoria virtual.

Seleccione una o más de una:

- ☒ a. El algoritmo óptimo no se puede implementar genéricamente y no tiene uso.
- ☐ b. El algoritmo NRU es una mejora del algoritmo clock.
- ☒ c. El algoritmo wsclock es una mejora del algoritmo working set.
- ☐ d. El algoritmo NRU es más fácil de implementar que el algoritmo working set.
- ☐ e. El algoritmo LRU es una simplificación del NFU y se ejecuta más rápido.



Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: El algoritmo NRU es más fácil de implementar que el algoritmo working set., El algoritmo wsclock es una mejora del algoritmo working set.

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea vTask1 que se ejecute de manera continua, informando este estado por el puerto serie.
- 2- vTask1 se debe crear en primer lugar.
- 3- Posea una tarea vTask2 que periódicamente cambie el estado del LED verde y se bloquee durante 750ms.

```
int main( void )
{
    [ ] ✓ ( [ ] ✓ , NULL, configMINIMAL_STACK_SIZE, NULL,
[ ] ✓ , NULL );
    [ ] ✓ ( [ ] ✓ , NULL, configMINIMAL_STACK_SIZE, NULL,
[ ] ✓ , NULL );
    [ ] ✓
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for( ;; ) {
        vPrintString( "Tarea continua en ejecución\r\n" );
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = [ ] ✓ ( 750 );

    for( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [ ] ✓ ( [ ] ✓ );
    }
}
```

tskIDLE_PRIORITY+3	vTaskStartScheduler();	vTaskDelete(NULL)	xTaskHandle
void vTask2		vTask1	vTask2
	tskIDLE_PRIORITY+2	xPeriod	for(;;)
vTaskDelay	xDelay		xTaskCreate
void vTask1	pvParameters	vTaskPrioritySet	&xLastWakeTime, xPeriod
vPrintString	xLastWakeTime	xTaskGetTickCount()	750
pdMS_TO_TICKS	vTaskDelayUntil		

Respuesta correcta

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea vTask1 que se ejecute de manera continua, informando este estado por el puerto serie.
- 2- vTask1 se debe crear en primer lugar.

3- Posea una tarea vTask2 que periódicamente cambie el estado del LED verde y se bloquee durante 750ms.

```
int main( void )
{
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+2], NULL );
    [xTaskCreate]( [vTask2], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+3], NULL );

    [vTaskStartScheduler();]
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for( ;; ) {
        vPrintString( "Tarea continua en ejecución\r\n" );
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 750 );

    for( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [vTaskDelay]( [xDelay] );
    }
}
```



Pregunta **10**

Parcialmente correcta

Puntúa 0,12 sobre 0,73

Indique lo correcto para RTOS expropiativos.

Seleccione una o más de una:

- ☐ a. Al seleccionar incorrectamente las prioridades de tareas foreground se puede presentar incoherencia de datos.
- ☐ b. La unidad básica de planificación son las tareas, y las mismas no deben bloquearse para evitar aumentar la latencia.
- ☐ c. Este tipo de RTOS permite disminuir el tiempo de latencia de foreground respecto a los RTOS no apropiativos.
- ☒ d. El uso correcto de semáforos o colas de mensaje permite evitar problemas de incoherencia. 
- ☒ e. Este tipo de planificadores hace un cambio de contexto cada vez que haya una tarea de foreground de igual o mas prioridad en estado listo (ready). 

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: El uso correcto de semáforos o colas de mensaje permite evitar problemas de incoherencia., Este tipo de RTOS permite disminuir el tiempo de latencia de foreground respecto a los RTOS no apropiativos.

Pregunta **11**

Parcialmente correcta

Puntúa 0,63 sobre 1,00

En el programa el hilo main crea 100 hilos.

- 1- Los hilos realizan una suma en la variable total y se sincronizan con un mutex.
- 2- El hilo main espera a que los hilos terminen, muestra el valor de la variable total y luego termina él.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int total, a[5]={0};
pthread_t h[100];
 ✓

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    pthread_mutex_lock(&m);
    for(t=0; t < 20000 ; t++){
         ✗
        total=total+numero;
         ✓
    }
     ✓
    pthread_exit(NULL);
}

int main() {
    int t;
    total=5000000;
    a[0]=10;
     ✓
    for(t=0; t< 100 ; t++){
         ✗
    }
    for(t=0; t< 100 ; t++){
         ✓
    }
    printf("Total= %d\n", total);
    pthread_exit(NULL);
}
/*
En consola vemos
Total=  ✗
*/
```

pthread_mutex_t m;

pthread_mutex_t m=PTHREAD_MUTEX_INITIALIZER;

2500000

pthread_mutex_unlock(&m);

25000000

pthread_detach(h[t],NULL)

//linea en blanco

250000000	pthread_join(h[1],NULL);
pthread_join(h[t],NULL);	pthread_create(&a[t], NULL, hilo0, (void *) (&a[t]))
	pthread_mutex_lock(&m);
pthread_create(&h[t], NULL, hilo1, (void *) (&a[0]));	

Respuesta parcialmente correcta.

Ha seleccionado correctamente 5.

La respuesta correcta es:

En el programa el hilo main crea 100 hilos.

1- Los hilos realizan una suma en la variable total y se sincronizan con un mutex.

2- El hilo main espera a que los hilos terminen, muestra el valor de la variable total y luego termina él.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int total, a[5]={0};
pthread_t h[100];
[pthread_mutex_t m=PTHREAD_MUTEX_INITIALIZER;]

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    pthread_mutex_lock(&m);
    for(t=0; t < 20000 ; t++){
        [//línea en blanco]
        total=total+numero;
        [//línea en blanco]
    }
    [pthread_mutex_unlock(&m);]
    pthread_exit(NULL);
}

int main() {
    int t;
    total=5000000;
    a[0]=10;
    [//línea en blanco]
    for(t=0; t< 100 ; t++){
        [pthread_create(&h[t], NULL, hilo1, (void *) (&a[0]))];]
    }
    for(t=0; t< 100 ; t++){
        [pthread_join(h[t],NULL);]
    }
    printf("Total= %d\n", total);
    pthread_exit(NULL);
}
/*
En consola vemos
Total= [25000000]
*/
```

Pregunta **12**

Parcialmente correcta

Puntúa 0,57 sobre 1,00

Un proceso posee 3 hilos (main, hilo0, hilo1). Los 3 hilos incrementan las variables total y total1.

- 1- El hilo main crea los hilos hilo1 e hilo0.
- 2- El hilo0 debe esperar al hilo1 antes de acceder a las variables total y total1,
- 3- El hilo main debe esperar al hilo1 antes de acceder a las variables total y total1.
- 4- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

```

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;

void *hilo0 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
     ✓
    for(t=0; t < 100000 ; t++){
        total=total+numero;
        total1=total1+1;
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
     ✓
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int j, a[5]={0};
    total=20000;
    total1=200000;
    a[0]=20;
    a[1]=10;
    a[3]=2;
     ✓
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
     ✗
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+1;
    }
     ✓
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total=  ✗ , Total1=  ✗
*/

```

 500000

 //linea en blanco

 pthread_join(h[2],NULL);

 pthread_yield();

 pthread_join(h[0],NULL);

 pthread_exit(NULL);

 pthread_detach(pthread_self());

 2250000

 pthread_mutexattr_init(&mtxattr);

 2230000

 5000000

 2220000

 pthread_join(h[1],NULL);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 4.

La respuesta correcta es:

Un proceso posee 3 hilos (main, hilo0, hilo1). Los 3 hilos incrementan las variables total y total1.

- 1- El hilo main crea los hilos hilo1 e hilo0.
- 2- El hilo0 debe esperar al hilo1 antes de acceder a las variables total y total1,
- 3- El hilo main debe esperar al hilo1 antes de acceder a las variables total y total1.
- 4- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;

void *hilo0 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    [pthread_join(h[1],NULL);]
    for(t=0; t < 100000 ; t++){
        total=total+numero;
        total1=total1+1;
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    [//linea en blanco]
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int j, a[5]={0};
    total=20000;
    total1=200000;
    a[0]=20;
    a[1]=10;
    a[3]=2;
    [//linea en blanco]
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    [pthread_join(h[0],NULL);]
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+1;
    }
    [//linea en blanco]
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total= [2230000] ,Total1= [500000]
*/
```


Comenzado el	jueves, 24 de junio de 2021, 19:00
Estado	Finalizado
Finalizado en	jueves, 24 de junio de 2021, 19:51
Tiempo empleado	51 minutos 7 segundos
Puntos	5,44/10,00
Calificación	5,44 de 10,00 (54%)


Pregunta **1**

Parcialmente correcta

Puntúa 0,36 sobre 0,71

Indique lo correcto referido a exclusión mutua del evento A respecto del evento B.

Seleccione una o más de una:

- ☐ a. Para forzar la situación de exclusión mutua, la única alternativa es usar un mutex inicializado en 1.
- ☐ b. El evento A y el evento B deben ejecutarse de manera simultánea.
- ☐ c. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un mutex para cada uno, ambos inicialmente en 1.
- ☐ d. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar barrier, inicialmente en 2.
- ☒ e. El evento A no debe ocurrir en el mismo momento que el evento B. 
- ☐ f. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un semáforo, inicialmente en 1.
- ☐ g. El evento A debe esperar la ocurrencia del evento B, e inversamente.
- ☐ h. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un mutex, inicialmente en 0.

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un semáforo, inicialmente en 1., El evento A no debe ocurrir en el mismo momento que el evento B.

Pregunta **2**

Parcialmente correcta

Puntúa 0,36 sobre 0,72

Seleccione lo correcto para implantación de RTS con procesamiento secuencial.

Seleccione una o más de una:

- ☐ a. Los bucles scan pueden tener problemas de incoherencia si comparten datos entre las tareas.
- ☒ b. Si se necesita un período de muestreo (T_s) determinado, se adiciona un timer y una tarea, siempre que T_s sea menor al tiempo que demoran las tareas que componen el bucle. ✓
- ☐ c. En caso que una tarea del bucle con periodo mayor a T_s y tiempo de ejecución largo, es posible dividirla en dos o más, para que se cumpla el T_s .
- ☐ d. Para tener determinismo, las tareas deben bloquearse esperando eventos externos asíncronos.
- ☐ e. La latencia máxima en un bucle siempre es menor al tiempo de muestreo, T_s .
- ☐ f. En caso que una tarea necesite un tiempo de muestreo distinto de T_s , se puede lograr con el uso de un timer.

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Si se necesita un período de muestreo (T_s) determinado, se adiciona un timer y una tarea, siempre que T_s sea menor al tiempo que demoran las tareas que componen el bucle., En caso que una tarea del bucle con periodo mayor a T_s y tiempo de ejecución largo, es posible dividirla en dos o más, para que se cumpla el T_s .

Pregunta **3**

Parcialmente correcta

Puntúa 0,75 sobre 1,00

En el programa el hilo main crea 100 hilos.

- 1- Los hilos realizan una suma en la variable total y se sincronizan con un mutex.
- 2- El hilo main espera a que los hilos terminen, muestra el valor de la variable total y luego termina él.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int total, a[5]={0};
pthread_t h[100];

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    pthread_mutex_lock(&m);
    for(t=0; t < 20000 ; t++){
        total=total+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int t;
    total=5000000;
    a[0]=10;
    for(t=0; t< 100 ; t++){
    }
    for(t=0; t< 100 ; t++){
    }
    printf("Total= %d\n", total);
    pthread_exit(NULL);
}
/*
En consola vemos
Total=
*/
```

pthread_mutex_t m=PTHREAD_MUTEX_INITIALIZER;

25000000

//linea en blanco

250000000

pthread_join(h[1],NULL);

pthread_mutex_lock(&m);

2500000

pthread_create(&a[t], NULL, hilo0, (void *) (&a[t]))

pthread_mutex_unlock(&m);

pthread_mutex_t m;

pthread_detach(h[t],NULL)

pthread_create(&h[t], NULL, hilo1, (void *) (&a[0]));

pthread_join(h[t],NULL);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 6.

La respuesta correcta es:

En el programa el hilo main crea 100 hilos.

1- Los hilos realizan una suma en la variable total y se sincronizan con un mutex.

2- El hilo main espera a que los hilos terminen, muestra el valor de la variable total y luego termina él.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int total, a[5]={0};
pthread_t h[100];
[pthread_mutex_t m=PTHREAD_MUTEX_INITIALIZER;]

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    pthread_mutex_lock(&m);
    for(t=0; t < 20000 ; t++){
        [//línea en blanco]
        total=total+numero;
        [//línea en blanco]
    }
    [pthread_mutex_unlock(&m);]
    pthread_exit(NULL);
}

int main() {
    int t;
    total=5000000;
    a[0]=10;
    [//línea en blanco]
    for(t=0; t< 100 ; t++){
        [pthread_create(&h[t], NULL, hilo1, (void *) (&a[0]));]
    }
    for(t=0; t< 100 ; t++){
        [pthread_join(h[t],NULL);]
    }
    printf("Total= %d\n", total);
    pthread_exit(NULL);
}
/*
En consola vemos
Total= [25000000]
*/
```

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe ejecutarse una única vez y debe ser la primera en ejecutarse.
- 2- vTask2 debe ser la primer tarea en crearse.
- 3- vTask2 debe cambiar el estado del LED verde y bloquearse durante 300ms.

```
int main( void )
{
    [ ] ✓ ( [ ] ✓ , NULL, configMINIMAL_STACK_SIZE, NULL,
[ ] ✓ , NULL );
    [ ] ✓ ( [ ] ✓ , NULL, configMINIMAL_STACK_SIZE, NULL,
[ ] ✓ , NULL );
    [ ] ✗ ;
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for ( ;; ) {
        vPrintString( "Ejecución única\r\n" );
        [ ] ✓ ;
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = [ ] ✓ ( 300 );

    for ( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [ ] ✓ ( [ ] ✓ );
    }
}
```

vTask1	xDelay	xTaskGetTickCount()	&xLastWakeTime, xPeriod
vPrintString	300	xLastWakeTime	
vTask2	void vTask2	vTaskDelayUntil	for(;;)
	tskIDLE_PRIORITY+1	xTaskCreate	vTaskStartScheduler()
xPeriod	pdMS_TO_TICKS	vTaskDelay	vTaskPrioritySet
pvParameters	void vTask1	tskIDLE_PRIORITY+2	vTaskDelete(NULL)
xTaskHandle			

Respuesta parcialmente correcta.

Ha seleccionado correctamente 10.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe ejecutarse una única vez y debe ser la primera en ejecutarse.
- 2- vTask2 debe ser la primer tarea en crearse.
- 3- vTask2 debe cambiar el estado del LED verde y bloquearse durante 300ms.

```
int main( void )
{
    [xTaskCreate]( [vTask2], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+1], NULL );
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+2], NULL );

    [vTaskStartScheduler()];
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for ( ;; ) {
        vPrintString( "Ejecución única\r\n" );
        [vTaskDelete( NULL )];
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 300 );

    for ( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [vTaskDelay]( [xDelay] );
    }
}
```

Pregunta **5**

Correcta

Puntúa 0,71 sobre 0,71

Seleccione lo correcto respecto algoritmos de ubicación de memoria, para gestión de memoria con particiones variables.

Seleccione una o más de una:

- ☐ a. El peor ajuste es más rápido para asignar memoria que el mejor ajuste.
- ☐ b. El mejor ajuste tiene rendimiento mejor que el primer ajuste.
- ☒ c. El mejor ajuste provoca más desperdicio de memoria que el primer ajuste.
- ☒ d. El primer ajuste es más rápido para asignar memoria que el peor ajuste.
- ☐ e. El primer ajuste tiene rendimiento algo peor que el siguiente ajuste.



Respuesta correcta

Las respuestas correctas son: El mejor ajuste provoca más desperdicio de memoria que el primer ajuste., El primer ajuste es más rápido para asignar memoria que el peor ajuste.

Pregunta **6**

Incorrecta

Puntúa 0,00 sobre 0,73

Seleccione lo correcto respecto a ISR.

- ☐ a. La manera de evitar conmutación de tareas dentro de las ISR es agregar una argumento a las llamadas a sistema del RTOS con un timeout.
- ☒ b. Si la ISR invoca una funcion del RTOS que pone una tarea prioritaria activa, se debe conmutar a esa tarea. ✗
- ☐ c. Las ISR no pueden invocar a funciones del RTOS.
- ☐ d. La manera de evitar conmutación de tareas dentro de las ISR es deshabilitar las interrupciones.
- ☐ e. Las ISR no deben invocar a funciones del RTOS que conmuten de tareas.

Respuesta incorrecta.

La respuesta correcta es: Las ISR no deben invocar a funciones del RTOS que conmuten de tareas.

ENUNCIADO

Modifique el archivo padre.c para que reciba del proceso hijo 3 caracteres mediante la cola de mensajes declarada como MQ_PATH y los imprima por consola.

El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr. Se recomienda que el padre elimine la cola de mensajes luego de leer el mensaje enviado por el proceso hijo.

El proceso hijo recibe como parámetro la variable mqd. Escribe en la cola de mensaje 3 caracteres y cierra la cola de mensaje. Luego de realizar estas acciones, escribe por consola los siguientes mensajes de confirmación:

Hijo en ejecucion...

Hijo: mensaje enviado.

Hijo: cola de mensajes cerrada.

PASOS A SEGUIR

1) Descargue los siguientes 2 archivos en un mismo directorio:

1.a) Código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/5WnAqmnscswdASWQ>

1.b) Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/FncdSNegcP9JTKq>

2) Complete el archivo padre.c. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr.

3) Copiar y pegar en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4) El binario generado por el comando anterior se ejecuta en consola con:

```
./padre
```

5) Luego que el proceso padre reciba los 3 caracteres los debe imprimir por consola:

```
XXX
```

RESPUESTA DEL EJERCICIO

Copie el número de 3 cifras XXX en la casilla de abajo. Este número es la respuesta del ejercicio.

Respuesta:



La respuesta correcta es: 241

Pregunta **8**

Parcialmente correcta

Puntúa 0,47 sobre 0,71

Seleccione lo verdadero respecto a segmentación para memoria virtual.

Seleccione una o más de una:

- ☐ a. Para acceder a una dirección física solo es necesario indicar el nombre del segmento y el offset.
- ☒ b. Cada segmento constituye un espacio de direcciones independiente. ✓
- ☒ c. Desacopla la relación 1 a 1 entre el bus de direcciones y la memoria física. ✓
- ☐ d. El objetivo principal es ejecutar programas más grandes que la memoria física.
- ☐ e. La memoria virtual se divide en segmentos independientes de igual tamaño y potencia de 2.

Respuesta parcialmente correcta.

Ha seleccionado correctamente 2.

Las respuestas correctas son: Cada segmento constituye un espacio de direcciones independiente., Desacopla la relación 1 a 1 entre el bus de direcciones y la memoria física., Para acceder a una dirección física solo es necesario indicar el nombre del segmento y el offset.

Pregunta **9**

Correcta

Puntúa 0,71 sobre 0,71

Indique lo correcto para el algoritmo de planificación Shortest Remaining Time Next.

Seleccione una o más de una:

- ☒ a. Se debe conocer de antemano el tiempo de ejecución de los trabajos. ✓
- ☐ b. Se asigna un quantum máximo de ejecución.
- ☐ c. Con este algoritmo se obtiene el menor tiempo de respuesta.
- ☒ d. Favorece la ejecución de tareas cortas. ✓
- ☒ e. Es la versión expropiativa de shortest job first. ✓
- ☐ f. Se debe contar con todas las tareas en el mismo momento.

Respuesta correcta

Las respuestas correctas son: Es la versión expropiativa de shortest job first., Se debe conocer de antemano el tiempo de ejecución de los trabajos., Favorece la ejecución de tareas cortas.

Pregunta **10**

Parcialmente correcta

Puntúa 0,17 sobre 1,00

Dos procesos relacionados se comunican por medio de una cola de mensajes.

- 1- El proceso padre crea la cola de mensajes.
- 2- El proceso padre crea al proceso hijo.
- 3- El proceso padre escribe un mensaje en la cola de mensajes, espera a que el proceso hijo termine y luego terminar él.
- 4- El proceso hijo lee un mensaje de la cola de mensajes, muestra lo leído.
- 5- El proceso hijo elimina la cola de mensajes y termina.

Completar

```
#include <mqueue.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>

#define AAAA "Parcial TD III"
#define MQTD3 "/TD3"

char b[20];
mqd_t m;
struct mq_attr at;

int main() {
    at.mq_msgsize = sizeof(b);
    at.mq_maxmsg = 3;
     ✖
    if (fork() > 0) {
         ✖
        mq_send(m, AAAA, strlen(AAAA)+1, 1);
        wait(NULL);
         ✔
        mq_close(m);
        exit(0);
    }
     ✖
     ✖
    printf("%s\n", b);
    mq_close(m);
     ✖
    exit(0);
}
```

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

La respuesta correcta es:

Dos procesos relacionados se comunican por medio de una cola de mensajes.

- 1- El proceso padre crea la cola de mensajes.
- 2- El proceso padre crea al proceso hijo.
- 3- El proceso padre escribe un mensaje en la cola de mensajes, espera a que el proceso hijo termine y luego terminar él.
- 4- El proceso hijo lee un mensaje de la cola de mensajes, muestra lo leído.
- 5- El proceso hijo elimina la cola de mensajes y termina.

Completar

```
#include <mqueue.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>

#define AAAA "Parcial TD III"
#define MQTD3 "/TD3"

char b[20];
mqd_t m;
struct mq_attr at;

int main() {
    at.mq_msgsize = sizeof(b);
    at.mq_maxmsg = 3;
    [m = mq_open(MQTD3, O_RDWR | O_CREAT, 0777, &at);]
    if (fork() > 0) {
        [//linea en blanco]
        mq_send(m, AAAA, strlen(AAAA)+1, 1);
        wait(NULL);
        [//linea en blanco]
        mq_close(m);
        exit(0);
    }
    [mq_getattr(m, &at);]
    [mq_receive(m, b, at.mq_msgsize, 0);]
    printf("%s\n", b);
    mq_close(m);
    [mq_unlink(MQTD3);]
    exit(0);
}
```

Pregunta **11**

Incorrecta

Puntúa 0,00 sobre 0,71

Indique cual de las siguientes afirmaciones son verdaderas cuando se ejecuta el comando `pthread_create()`.

Seleccione una o más de una:

- ☐ a. Es posible configurar el tamaño de stack y la prioridad del hilo en el segundo argumento de la llamada.
- ☒ b. Si no hubo error, retorna el TID (Thread ID) del hilo que crea. ✗
- ☐ c. Si no hubo error, almacena en un buffer indicado por el primer argumento el TID,
- ☐ d. Si la creación es exitosa, el hilo ejecuta la función indicada como tercer argumento.
- ☒ e. Si la creación del hilo no es exitosa, retorna el número de error. ✓
- ☐ f. Si hubo error, retorna -1 y no es exitosa la creación del hilo.

Respuesta incorrecta.

Las respuestas correctas son: Es posible configurar el tamaño de stack y la prioridad del hilo en el segundo argumento de la llamada., Si la creación es exitosa, el hilo ejecuta la función indicada como tercer argumento., Si la creación del hilo no es exitosa, retorna el número de error., Si no hubo error, almacena en un buffer indicado por el primer argumento el TID,

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Ambas tareas compiten por el uso del puerto serie, un recurso crítico.
- 2- vTask1 espera indefinidamente la liberación del recurso.
- 3- vTask2 espera 1 segundo, si no se libera el recurso cambia de estado el LED rojo.

```
SemaphoreHandle_t xMutex;  
SemaphoreHandle_t xBinarySemaphore;  
int main( void )  
{  
    [ ] ✓ = [ ] ✓ ;  
  
    /*Creación de tareas, habilitación de interrupciones, inicio planificador*/  
  
    for( ;; );  
    return 0;  
}  
  
void vTask1 ( void *pvParameters )  
{  
    const TickType_t xDelay = [ ] ✓ ( 1000 );  
  
    for ( ;; ) {  
        if( [ ] ✓ ( [ ] ✓ ) == pdTRUE) {  
            vPrintString( "Acceso critico vTask1\r\n" );  
            [ ] ✓ ( [ ] ✓ );  
        } else Board_LED_Toggle(5); //LED verde  
    }  
}  
  
void vTask2 ( void *pvParameters )  
{  
    const TickType_t xDelay = [ ] ✓ ( 1000 );  
  
    for ( ;; ) {  
        if( [ ] ✓ ( [ ] ✓ ) == pdTRUE) {  
            vPrintString( "Acceso critico vTask2\r\n" );  
            [ ] ✓ ( [ ] ✓ );  
        } else Board_LED_Toggle(4); //LED rojo  
    }  
}
```

xBinarySemaphore, xDelay

xSemaphoreCreateMutex()

xMutex, portMAX_DELAY

xSemaphoreGive

xMutex, xDelay

xDelay

pdMS_TO_TICKS

xSemaphoreTakeFromISR

xSemaphoreTake

xHigherPriorityTaskWoken

xBinarySemaphore

xMutex

xSemaphoreCreateBinary()

xBinarySemaphore, portMAX_DELAY

portYIELD_FROM_ISR

xSemaphoreGiveFromISR

Respuesta correcta

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Ambas tareas compiten por el uso del puerto serie, un recurso crítico.
- 2- vTask1 espera indefinidamente la liberación del recurso.
- 3- vTask2 espera 1 segundo, si no se libera el recurso cambia de estado el LED rojo.

```
SemaphoreHandle_t xMutex;
SemaphoreHandle_t xBinarySemaphore;
int main( void )
{
    [xMutex] = [xSemaphoreCreateMutex()];

    /*Creación de tareas, habilitación de interrupciones, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );
    for ( ;; ) {
        if( [xSemaphoreTake]( [xMutex, portMAX_DELAY] ) == pdTRUE) {
            vPrintString( "Acceso critico vTask1\r\n" );
            [xSemaphoreGive]( [xMutex] );
        } else Board_LED_Toggle(5); //LED verde
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        if( [xSemaphoreTake]( [xMutex, xDelay] ) == pdTRUE) {
            vPrintString( "Acceso critico vTask2\r\n" );
            [xSemaphoreGive]( [xMutex] );
        } else Board_LED_Toggle(4); //LED rojo
    }
}
```

[◀ Recuperatorio Parcial 1](#)

Ir a...

Comenzado el	jueves, 24 de junio de 2021, 19:00
Estado	Finalizado
Finalizado en	jueves, 24 de junio de 2021, 19:47
Tiempo empleado	47 minutos 20 segundos
Puntos	6,85/10,00
Calificación	6,84 de 10,00 (68%)

Pregunta **1**

Incorrecta

Puntúa 0,00 sobre 0,71

Indique lo correcto respecto a la planificación de procesos.

- ☐ a. Al crearse hilos en espacio usuario, se deben modificar los planificadores para contemplarlos.
- ☐ b. La necesidad de los planificadores surge para poder tener en memoria más de un proceso a la vez.
- ☐ c. Los planificadores expropiativos solo se ejecutan cuando el proceso en estado RUN ejecuta una syscall.
- ☒ d. Con un planificador no expropiativo se ven beneficiadas las tareas I/O bounded. ✗
- ☒ e. Cuanto menor sea el quantum que utilizan los planificadores, más eficiente será el sistema operativo. ✗

Respuesta incorrecta.

La respuesta correcta es: La necesidad de los planificadores surge para poder tener en memoria más de un proceso a la vez.

Pregunta **2**

Parcialmente correcta

Puntúa 0,12 sobre 0,72

Indique lo correcto respecto a RTS.

Seleccione una o más de una:

- ☐ a. En el nivel de software interface de usuario, no hay restricciones temporales estrictas.
- ☐ b. El uso de un RTOS permite asegurar el determinismo para un RTS.
- ☐ c. Los algoritmos de control son el nivel de software mas cercano al hardware.
- ☒ d. El periodo de muestreo (Ts) se refiere al periodo necesario para la adquisición de datos. ✗
- ☒ e. Los algoritmos de supervisión se ejecutan con periodos generalmente mayores a los algoritmos de control. ✓

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Los algoritmos de supervisión se ejecutan con periodos generalmente mayores a los algoritmos de control., En el nivel de software interface de usuario, no hay restricciones temporales estrictas.

Pregunta **3**

Parcialmente correcta

Puntúa 0,24 sobre 0,71

Cuál de las siguientes afirmaciones es verdadera respecto al uso de hilos.

Seleccione una o más de una:

- ☒ a. Es mas rápida la creación y destrucción de hilos respecto de subrutinas.
- ☒ b. Puede mejorar la velocidad de ejecución en sistemas multiprocesador.
- ☒ c. La ejecución de hilos es separada y los recursos son compartidos.
- ☒ d. Comparten el registro de estado.
- ☐ e. Nunca es necesaria la sincronización de hilos.



Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

Las respuestas correctas son: La ejecución de hilos es separada y los recursos son compartidos., Puede mejorar la velocidad de ejecución en sistemas multiprocesador.

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Ambas tareas compiten por el uso del puerto serie, un recurso crítico.
- 2- vTask1 espera indefinidamente la liberación del recurso.
- 3- vTask2 espera 1 segundo, si no se libera el recurso cambia de estado el LED rojo.

```
SemaphoreHandle_t xMutex;  
SemaphoreHandle_t xBinarySemaphore;  
int main( void )  
{  
    [ ] ✓ = [ ] ✓ ;  
  
    /*Creación de tareas, habilitación de interrupciones, inicio planificador*/  
  
    for( ;; );  
    return 0;  
}  
  
void vTask1 ( void *pvParameters )  
{  
    const TickType_t xDelay = [ ] ✓ ( 1000 );  
  
    for ( ;; ) {  
        if( [ ] ✓ ( [ ] ✓ ) == pdTRUE) {  
            vPrintString( "Acceso crítico vTask1\r\n" );  
            [ ] ✓ ( [ ] ✓ );  
        } else Board_LED_Toggle(5); //LED verde  
    }  
}  
  
void vTask2 ( void *pvParameters )  
{  
    const TickType_t xDelay = [ ] ✓ ( 1000 );  
  
    for ( ;; ) {  
        if( [ ] ✓ ( [ ] ✓ ) == pdTRUE) {  
            vPrintString( "Acceso crítico vTask2\r\n" );  
            [ ] ✓ ( [ ] ✓ );  
        } else Board_LED_Toggle(4); //LED rojo  
    }  
}
```

	xSemaphoreGive	portYIELD_FROM_ISR
xSemaphoreCreateBinary()	xBinarySemaphore, xDelay	xBinarySemaphore
xBinarySemaphore, portMAX_DELAY		xSemaphoreTake
xSemaphoreGiveFromISR	xSemaphoreTakeFromISR	xMutex, xDelay
xDelay		pdMS_TO_TICKS
	xMutex, portMAX_DELAY	
xSemaphoreCreateMutex()		xMutex
xHigherPriorityTaskWoken		

Respuesta correcta

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Ambas tareas compiten por el uso del puerto serie, un recurso crítico.
- 2- vTask1 espera indefinidamente la liberación del recurso.
- 3- vTask2 espera 1 segundo, si no se libera el recurso cambia de estado el LED rojo.

```
SemaphoreHandle_t xMutex;
SemaphoreHandle_t xBinarySemaphore;
int main( void )
{
    [xMutex] = [xSemaphoreCreateMutex()];

    /*Creación de tareas, habilitación de interrupciones, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );
    for ( ;; ) {
        if( [xSemaphoreTake]( [xMutex, portMAX_DELAY] ) == pdTRUE) {
            vPrintString( "Acceso crítico vTask1\r\n" );
            [xSemaphoreGive]( [xMutex] );
        } else Board_LED_Toggle(5); //LED verde
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        if( [xSemaphoreTake]( [xMutex, xDelay] ) == pdTRUE) {
            vPrintString( "Acceso critico vTask2\r\n" );
            [xSemaphoreGive]( [xMutex] );
        } else Board_LED_Toggle(4); //LED rojo
    }
}
```

ENUNCIADO

Modifique el archivo padre.c para que reciba del proceso hijo 3 caracteres mediante la cola de mensajes declarada como MQ_PATH y los imprima por consola.

El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr. Se recomienda que el padre elimine la cola de mensajes luego de leer el mensaje enviado por el proceso hijo.

El proceso hijo recibe como parámetro la variable mqd. Escribe en la cola de mensaje 3 caracteres y cierra la cola de mensaje. Luego de realizar estas acciones, escribe por consola los siguientes mensajes de confirmación:

Hijo en ejecucion...

Hijo: mensaje enviado.

Hijo: cola de mensajes cerrada.

PASOS A SEGUIR

1) Descargue los siguientes 2 archivos en un mismo directorio:

1.a) Código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/5WnAqmnscswdASWQ>

1.b) Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/EPTxT9N7oPqWBBf>

2) Complete el archivo padre.c. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr.

3) Copiar y pegar en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4) El binario generado por el comando anterior se ejecuta en consola con:

```
./padre
```

5) Luego que el proceso padre reciba los 3 caracteres los debe imprimir por consola:

```
XXX
```

RESPUESTA DEL EJERCICIO

Copie el número de 3 cifras XXX en la casilla de abajo. Este número es la respuesta del ejercicio.



Respuesta:



La respuesta correcta es: 587

Indique lo correcto referido a concurrencia del evento A respecto del evento B.

Seleccione una o más de una:

- ☐ a. Para forzar la situación de concurrencia (para dos eventos) se puede usar barrier, inicialmente en 2.
- ☒ b. El evento A y el evento B pueden ejecutarse simultáneamente en caso de usar sistemas multitarea. 
- ☐ c. El evento A no debe ocurrir en el mismo momento que el evento B.
- ☐ d. Para forzar la situación de concurrencia (para dos eventos) se puede usar un mutex para cada uno, ambos inicialmente en 1.
- ☐ e. Para forzar la situación de concurrencia (para dos eventos) se puede usar un semáforo, inicialmente en 1.
- ☒ f. No es necesario forzar la situación de concurrencia, ni usar mutex, barrier o semáforos. 
- ☐ g. Para forzar la situación de concurrencia (para dos eventos) se puede usar un mutex, inicialmente en 0.
- ☐ h. El evento A debe esperar la ocurrencia del evento B, e inversamente.

Respuesta correcta

Las respuestas correctas son: El evento A y el evento B pueden ejecutarse simultáneamente en caso de usar sistemas multitarea., No es necesario forzar la situación de concurrencia, ni usar mutex, barrier o semáforos.

Complete la función main() de la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Cree dos tareas: vTask1 y vTask2.
- 2- vTask1, creada en primer lugar, periódicamente modifica la prioridad de vTask2 para que esta pueda ejecutarse.

TaskHandle_t xTaskHandle;

int main(void)

{

✓ (✓ , NULL, configMINIMAL_STACK_SIZE, NULL,

✓ , ✓);

✓ (✓ , NULL, configMINIMAL_STACK_SIZE, NULL,

✓ , ✓);

✓ ;

for(;;);

return 0;

}

void vTask1 (void *pvParameters)

{ ... }

void vTask2 (void *pvParameters)

{ ... }

vTask1

vTaskStartScheduler()

pdMS_TO_TICKS

&xLastWakeTime, xTaskHandle

tskIDLE_PRIORITY+1

xDelay

vPrintString

xTaskCreate

tskIDLE_PRIORITY+2

pvParameters

xPeriod

vTask2

xLastWakeTime

xTaskHandle

tskIDLE_PRIORITY+3

NULL

for(;;)

vTaskDelayUntil

&xTaskHandle

vTaskDelete(NULL)

xTaskGetTickCount()

vTaskDelay

vTaskPrioritySet

void vTask2

void vTask1

Respuesta correcta

La respuesta correcta es:

Complete la función main() de la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Cree dos tareas: vTask1 y vTask2.
- 2- vTask1, creada en primer lugar, periódicamente modifica la prioridad de vTask2 para que esta pueda ejecutarse.

```
TaskHandle_t xTaskHandle;
int main( void )
{
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+2], [NULL]);
    [xTaskCreate]( [vTask2], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+1], [&xTaskHandle]);

    [vTaskStartScheduler()];
    for( ;; );
    return 0;
}
```

```
void vTask1 ( void *pvParameters )  
{ ... }  
  
void vTask2 ( void *pvParameters )  
{ ... }
```

Pregunta 8

Parcialmente correcta

Puntúa 0,83 sobre 1,00

Un proceso posee 3 hilos: main, hilo0, hilo1. Los 3 hilos incrementan las variables total y total1.

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- El hilo1 debe esperar al hilo0 antes de acceder a las variables total y total1,
- 3- El hilo main debe esperar al hilo1 antes de acceder a las variables total y total1.
- 4- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;

void *hilo0 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
     ✓
    for(t=0; t < 100000 ; t++){
        total=total+numero;
        total1=total1+1;
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
     ✓
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int j, a[5]={0};
    total=10000;
    total1=100000;
    a[0]=10;
    a[1]=2;
    a[3]=1;
     ✓
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
     ✓
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+1;
    }
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total=  ✗ ,Total1=  ✓
*/
```

			//linea en blanco			
			pthread_join(h[0],NULL);	320000	11200000	
			pthread_yield();	3200000	pthread_mutexattr_init(&mtxattr);	
			pthread_join(h[1],NULL);	pthread_detach(pthread_self());	1210000	
			1120000	pthread_join(h[2],NULL);	pthread_exit(NULL);	

Respuesta parcialmente correcta.

Ha seleccionado correctamente 5.

La respuesta correcta es:

Un proceso posee 3 hilos: main, hilo0, hilo1. Los 3 hilos incrementan las variables total y total1.

1- El hilo main crea los hilos hilo0 e hilo1.

2- El hilo1 debe esperar al hilo0 antes de acceder a las variables total y total1,

3- El hilo main debe esperar al hilo1 antes de acceder a las variables total y total1.

4- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:


```

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;

void *hilo0 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    [//linea en blanco]
    for(t=0; t < 100000 ; t++){
        total=total+numero;
        total1=total1+1;
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    [pthread_join(h[0],NULL);]
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int j, a[5]={0};
    total=10000;
    total1=100000;
    a[0]=10;
    a[1]=2;
    a[3]=1;
    [//linea en blanco]
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
    [pthread_join(h[1],NULL);]
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+1;
    }
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total= [1120000] ,Total1= [320000]
*/

```

Pregunta **9**

Parcialmente correcta

Puntúa 0,55 sobre 0,73

Indique lo correcto referido a RTOS no apropiativos.

Seleccione una o más de una:

- ☒ a. Es más fácil la gestión de ISR en los RTOS no apropiativos que en los RTOS apropiativos.
- ☐ b. En este tipo de RTOS no existen los problemas de incoherencia de datos.
- ☐ c. Permiten gestionar mejor las tareas de background que en los sistemas foreground/background.
- ☒ d. Tienen mayor latencia las tareas de foreground que en en los RTOS apropiativos.
- ☐ e. Es más eficiente que un sistema foreground/background.



Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

La respuesta correcta es: Tienen mayor latencia las tareas de foreground que en en los RTOS apropiativos.

Pregunta **10**

Parcialmente correcta

Puntúa 0,57 sobre 1,00

Un proceso posee 3 hilos (main, hilo0, hilo1).

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- Los hilos hilo1 e hilo2 incrementan las variables total y total1.
- 3- El hilo main incrementa la variable total.
- 4- El hilo main espera a que los hilos terminen y muestra el resultado de total y total1.

Completar:

```

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;
pthread_mutex_t mtx0=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mtx1=PTHREAD_MUTEX_INITIALIZER;

void * hilo1() {
    int t;
     ✓
     ✓
    for(t=0; t < 100000 ; t++){
        total=total+2;
        total1=total1+1;
    }
    pthread_mutex_unlock(&mtx1);
    pthread_mutex_unlock(&mtx0);
    pthread_exit(NULL);
}

void * hilo2() {
    int t;
    pthread_mutex_lock(&mtx0);
    pthread_mutex_lock(&mtx1);
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+2;
    }
    pthread_mutex_unlock(&mtx1);
    pthread_mutex_unlock(&mtx0);
    pthread_exit(NULL);
}

int main() {
    int t;
    total=20000;
    total1=1000;
    pthread_create (&h[0], NULL, hilo1, NULL);
    pthread_create (&h[1], NULL, hilo2, NULL);
     ✓
    for(t=0; t < 1000000 ; t++){
         ✓
        total=total+1;
        total=total+1;
    }
    pthread_mutex_unlock(&mtx0);
    pthread_join(h[0],NULL);
     ✗
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total=  ✗ ,Total1=  ✗
*/

```

pthread_join(h[0],NULL);

pthread_mutex_lock(&mtx1);

2130000

2300000

pthread_mutex_lock(&mtx0);

2230000

//linea en blanco

22300000

pthread_mutex_unlock(&mtx1);

1210000

2200000

pthread_join(h[1],NULL);

pthread_mutex_unlock(&mtx0);

121000

Ha seleccionado correctamente 4.

La respuesta correcta es:

Un proceso posee 3 hilos (main, hilo0, hilo1).

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- Los hilos hilo1 e hilo2 incrementan las variables total y total1.
- 3- El hilo main incrementa la variable total.
- 4- El hilo main espera a que los hilos terminen y muestra el resultado de total y total1.

Completar:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;
pthread_mutex_t mtx0=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mtx1=PTHREAD_MUTEX_INITIALIZER;

void * hilo1() {
    int t;
    [pthread_mutex_lock(&mtx0);]
    [pthread_mutex_lock(&mtx1);]
    for(t=0; t < 100000 ; t++){
        total=total+2;
        total1=total1+1;
    }
    pthread_mutex_unlock(&mtx1);
    pthread_mutex_unlock(&mtx0);
    pthread_exit(NULL);
}

void * hilo2() {
    int t;
    pthread_mutex_lock(&mtx0);
    pthread_mutex_lock(&mtx1);
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+2;
    }
    pthread_mutex_unlock(&mtx1);
    pthread_mutex_unlock(&mtx0);
    pthread_exit(NULL);
}

int main() {
    int t;
    total=20000;
    total1=1000;
    pthread_create (&h[0], NULL, hilo1, NULL);
    pthread_create (&h[1], NULL, hilo2, NULL);
    [pthread_mutex_lock(&mtx0);]
    for(t=0; t < 1000000 ; t++){
        [//línea en blanco]
        total=total+1;
        total=total+1;
    }
    pthread_mutex_unlock(&mtx0);
    pthread_join(h[0],NULL);
    [pthread_join(h[1],NULL);]
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total= [2230000] ,Total1= [121000]
*/
```

Pregunta **11**

Correcta

Puntúa 0,71 sobre 0,71

Seleccione lo correcto respecto de gestión de memoria con particiones fijas.

Seleccione una o más de una:

- ☐ a. Todas las particiones deben ser del mismo tamaño para evitar la fragmentación externa.
- ☐ b. Puede existir fragmentación externa.
- ☐ c. Se utiliza la prioridad para asignar particiones con múltiples colas.
- ☐ d. Es conveniente hacer compactación de memoria, aunque es costosa computacionalmente.
- ☒ e. Para asignar particiones se pueden usar cola única o múltiples colas. ✓
- ☒ f. Puede existir fragmentación interna. ✓

Respuesta correcta

Las respuestas correctas son: Puede existir fragmentación interna., Para asignar particiones se pueden usar cola única o múltiples colas.

Pregunta **12**

Parcialmente correcta

Puntúa 0,12 sobre 0,71

Seleccione lo verdadero respecto a algoritmos de reemplazo de página para memoria virtual.

Seleccione una o más de una:

- ☐ a. El algoritmo NRU es una mejora del algoritmo clock.
- ☐ b. El algoritmo óptimo no se puede implementar genéricamente y no tiene uso.
- ☒ c. El algoritmo wsclock es una mejora del algoritmo working set. ✓
- ☐ d. El algoritmo NRU es más fácil de implementar que el algoritmo working set.
- ☒ e. El algoritmo LRU es una simplificación del NFU y se ejecuta más rápido. ✗

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: El algoritmo NRU es más fácil de implementar que el algoritmo working set., El algoritmo wsclock es una mejora del algoritmo working set.

◀ Recuperatorio Parcial 1

Ir a...



Comenzado el	jueves, 24 de junio de 2021, 19:01
Estado	Finalizado
Finalizado en	jueves, 24 de junio de 2021, 19:44
Tiempo empleado	43 minutos 30 segundos
Puntos	4,75/10,00
Calificación	4,75 de 10,00 (48%)

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea vTask1 que se ejecute de manera continua e informe este estado por el puerto serie.
- 2- vTask1 se debe crear en primer lugar.
- 3- Posea una tarea vTask2 que pase a estado "Lista" exactamente cada 750ms y cambie el estado del LED verde.

```
int main( void )
{
    xTaskCreate ( vTask1 , NULL, configMINIMAL_STACK_SIZE, NULL,
    tskIDLE_PRIORITY+3 , NULL );
    ( ) ( , NULL, configMINIMAL_STACK_SIZE, NULL,
    ( ) , NULL );
    ( )
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for( ;; ) {
        vPrintString( "Tarea continua en ejecución\r\n" );
    }
}

void vTask2 ( void *pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = ( 750 );
    xLastWakeTime = ( );
    for( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        ( ) ( );
    }
}

xPeriod &xLastWakeTime, xPeriod vTaskPrioritySet
vTaskStartScheduler(); xTaskGetTickCount() xDelay for( ;; )
void vTask2 vTaskDelete( NULL ) pvParameters
pdMS_TO_TICKS vTask2 void vTask1 xLastWakeTime
750 vTaskDelayUntil vPrintString tskIDLE_PRIORITY+2
xTaskHandle vTaskDelay
```

Respuesta parcialmente correcta.

Ha seleccionado correctamente 6.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea vTask1 que se ejecute de manera continua e informe este estado por el puerto serie.
- 2- vTask1 se debe crear en primer lugar.
- 3- Posea una tarea vTask2 que pase a estado "Lista" exactamente cada 750ms y cambie el estado del LED verde.

```
int main( void )
{
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+2], NULL );
    [xTaskCreate]( [vTask2], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+3], NULL );

    [vTaskStartScheduler();]

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for( ;; ) {
        vPrintString( "Tarea continua en ejecución\r\n" );
    }
}

void vTask2 ( void *pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = [pdMS_TO_TICKS]( 750 );

    xLastWakeTime = [xTaskGetTickCount()];
    for( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [vTaskDelayUntil]( [&xLastWakeTime, xPeriod] );
    }
}
```

Pregunta **2**

Incorrecta

Puntúa 0,00 sobre 0,71

Indique lo correcto para el algoritmo de planificación Round-Robin.

Seleccione una o más de una:

- ☐ a. Es un algoritmo equitativo. ✗
- ☒ b. Disminuye el tiempo de retorno. ✗
- ☒ c. Si a una tarea en ejecución se le termina el quantum, va al final de la FIFO. ✓
- ☐ d. No usa técnicas de envejecimiento.
- ☐ e. Los trabajos deben estar disponibles al mismo tiempo.
- ☒ f. Es un algoritmo no expropiativo. ✗

Respuesta incorrecta.

Las respuestas correctas son: No usa técnicas de envejecimiento., Si a una tarea en ejecución se le termina el quantum, va al final de la FIFO., Es un algoritmo equitativo.

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Ambas tareas compiten por el uso del puerto serie, un recurso crítico.
- 2- vTask1 espera indefinidamente la liberación del recurso.
- 3- vTask2 espera 1 segundo, si no se libera el recurso cambia de estado el LED rojo.

```
SemaphoreHandle_t xMutex;
SemaphoreHandle_t xBinarySemaphore;
int main( void )
{
    [ xMutex ] ✓ = [ xSemaphoreCreateMutex() ] ✓ ;

    /*Creación de tareas, habilitación de interrupciones, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    const TickType_t xDelay = [ pdMS_TO_TICKS ] ✓ ( 1000 );

    for ( ;; ) {
        if( [ xSemaphoreTake ] ✓ ( [ xMutex, portMAX_DELAY ] ✓ ) == pdTRUE) {
            vPrintString( "Acceso critico vTask1\r\n" );
            [ xSemaphoreGive ] ✓ ( [ ] ✓ );
        } else Board_LED_Toggle(5); //LED verde
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = [ ] ✓ ( 1000 );

    for ( ;; ) {
        if( [ ] ✓ ( [ ] ✓ ) == pdTRUE) {
            vPrintString( "Acceso critico vTask2\r\n" );
            [ ] ✓ ( [ ] ✓ );
        } else Board_LED_Toggle(4); //LED rojo
    }
}
```

xBinarySemaphore

xSemaphoreTakeFromISR

xHigherPriorityTaskWoken

xBinarySemaphore, portMAX_DELAY

xSemaphoreCreateBinary()

xSemaphoreGiveFromISR

xBinarySemaphore, xDelay

xDelay

portYIELD_FROM_ISR

xMutex, xDelay

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Ambas tareas compiten por el uso del puerto serie, un recurso crítico.
- 2- vTask1 espera indefinidamente la liberación del recurso.
- 3- vTask2 espera 1 segundo, si no se libera el recurso cambia de estado el LED rojo.

```
SemaphoreHandle_t xMutex;
SemaphoreHandle_t xBinarySemaphore;

int main( void )
{
    [xMutex] = [xSemaphoreCreateMutex()];

    /*Creación de tareas, habilitación de interrupciones, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );
    for ( ;; ) {
        if( [xSemaphoreTake]( [xMutex, portMAX_DELAY] ) == pdTRUE) {
            vPrintString( "Acceso critico vTask1\r\n" );
            [xSemaphoreGive]( [xMutex] );
        } else Board_LED_Toggle(5); //LED verde
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );
    for ( ;; ) {
        if( [xSemaphoreTake]( [xMutex, xDelay] ) == pdTRUE) {
            vPrintString( "Acceso critico vTask2\r\n" );
            [xSemaphoreGive]( [xMutex] );
        } else Board_LED_Toggle(4); //LED rojo
    }
}
```

Pregunta **4**

Parcialmente correcta

Puntúa 0,47 sobre 0,71

Seleccione lo verdadero respecto a algoritmos de reemplazo de página para memoria virtual.

Seleccione una o más de una:

- ☐ a. El algoritmo FIFO es el más usado por su bajo overhead.
- ☒ b. El algoritmo aging es una modificación en software del LRU. ✓
- ☐ c. El algoritmo wsclock es una mejora del algoritmo clock.
- ☒ d. El algoritmo óptimo es imposible de implementar, pero se usa con fines comparativos. ✓
- ☒ e. El algoritmo segunda no contempla la páginas que se usan muy frecuentemente. ✗

Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

Las respuestas correctas son: El algoritmo óptimo es imposible de implementar, pero se usa con fines comparativos., El algoritmo aging es una modificación en software del LRU.

Pregunta **5**

Correcta

Puntúa 0,72 sobre 0,72

Una lo que corresponda.

Un sistema background/foreground respecto de un scan loop	disminuye la latencia de las tareas de segundo plano.	✓
un RTOS expropiativo respecto a un RTOS no expropiativo	disminuye la latencia de las tareas de primer plano.	✓
Un RTOS no expropiativo	puede tener incoherencia solo si se comparten datos entre ISR y tareas de primer plano.	✓
Un sistema scan loop	tiene el mayor de los rendimientos.	✓

Respuesta correcta

La respuesta correcta es: Un sistema background/foreground respecto de un scan loop → disminuye la latencia de las tareas de segundo plano., un RTOS expropiativo respecto a un RTOS no expropiativo → disminuye la latencia de las tareas de primer plano., Un RTOS no expropiativo → puede tener incoherencia solo si se comparten datos entre ISR y tareas de primer plano., Un sistema scan loop → tiene el mayor de los rendimientos.

Pregunta **6**

Incorrecta

Puntúa 0,00 sobre 0,71

Seleccione lo correcto respecto de gestión de memoria con particiones fijas.

Seleccione una o más de una:

- ☐ a. Se utiliza la prioridad para asignar particiones con múltiples colas.
- ☒ b. Todas las particiones deben ser del mismo tamaño para evitar la fragmentación externa. ✗
- ☒ c. Para asignar particiones se pueden usar cola única o múltiples colas. ✓
- ☐ d. Puede existir fragmentación interna.
- ☐ e. Es conveniente hacer compactación de memoria, aunque es costosa computacionalmente.
- ☒ f. Puede existir fragmentación externa. ✗

Respuesta incorrecta.

Las respuestas correctas son: Puede existir fragmentación interna., Para asignar particiones se pueden usar cola única o múltiples colas.

Pregunta **7**

Parcialmente correcta

Puntúa 0,80 sobre 1,00

Dos procesos relacionados se comunican por medio de una cola de mensajes.

- 1- El proceso padre crea al proceso hijo.
- 2- El proceso padre crea la cola de mensajes.
- 3- El proceso padre escribe un mensaje en la cola de mensajes, espera a que el proceso hijo termine y luego terminar él.
- 4- El proceso hijo lee un mensaje de la cola de mensajes, muestra lo leído y termina.

Completar.

```
#include <mqueue.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>

#define BB "Parcial TD III"
#define SHMM "/TD3"

int rc,t;
char a[1024];
mqd_t m;
struct mq_attr at;
```

```
int main() {
    if (fork() == 0) {
        m = mq_open(SHMM, O_RDWR, 0);
        mq_getattr(m, &at);
        mq_receive(m, a, at.mq_msgsize, 0);
        printf("%s\n", a);
        exit(0);
    }
    at.mq_msgsize = sizeof(a);
    at.mq_maxmsg = 5;
    m = mq_open(SHMM, O_WRONLY, 0777, &at);
    mq_send(m, BB, strlen(BB)+1, 1);
    wait(NULL);
    exit(0);
}
```

//línea en blanco

m = mq_open(SHMM, O_WRONLY, 0);

mq_getattr(mqd, &attr.mq_msgsize);

m = mq_open(SHMM, O_RDWR | O_CREAT, 0777, &at);

mq_receive(m, BB, strlen(BB), 0);

mq_send(m, a, at.mq_msgsize, 0);

m = mq_open(SHMM, O_RDWR, 0777, &at);

mq_close(mqd);

m = mq_open(SHMM, O_RDWR, 0);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 4.

La respuesta correcta es:

Dos procesos relacionados se comunican por medio de una cola de mensajes.

- 1- El proceso padre crea al proceso hijo.
- 2- El proceso padre crea la cola de mensajes.
- 3- El proceso padre escribe un mensaje en la cola de mensajes, espera a que el proceso hijo termine y luego terminar él.
- 4- El proceso hijo lee un mensaje de la cola de mensajes, muestra lo leído y termina.

Completar.

```
#include <mqueue.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>
```

```
#define BB "Parcial TD III"
#define SHMM "/TD3"
```

```
int rc,t;
char a[1024];
mqd_t m;
struct mq_attr at;
```

```
int main() {
    if (fork() == 0) {
        m = mq_open(SHMM, O_RDWR, 0);
        [mq_getattr(m, &at);]
        [mq_receive(m, a, at.mq_msgsize, 0);]
        [printf("%s\n", a);]
        exit(0);
    }
    at.mq_msgsize = sizeof(a);
    at.mq_maxmsg = 5;
    [m = mq_open(SHMM, O_RDWR | O_CREAT, 0777, &at);]
    [mq_send(m, BB, strlen(BB)+1, 1);]
    wait(NULL);
    exit(0);
}
```

Indique lo correcto referido a exclusión mutua del evento A respecto del evento B.

Seleccione una o más de una:

- ☒ a. El evento A no debe ocurrir en el mismo momento que el evento B. ✓
- ☐ b. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un mutex, inicialmente en 0.
- ☒ c. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar barrier, inicialmente en 2. ✗
- ☐ d. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un semáforo, inicialmente en 1.
- ☐ e. Para forzar la situación de exclusión mutua, la única alternativa es usar un mutex inicializado en 1.
- ☐ f. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un mutex para cada uno, ambos inicialmente en 1.
- ☒ g. El evento A debe esperar la ocurrencia del evento B, e inversamente. ✗
- ☐ h. El evento A y el evento B deben ejecutarse de manera simultánea.

Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

Las respuestas correctas son: Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un semáforo, inicialmente en 1., El evento A no debe ocurrir en el mismo momento que el evento B.

Una lo que corresponda referido a multi threading.

- | | | |
|---|---|---|
| En la implementación de hilos de manera híbrida | cada hilo de nivel kernel tiene algún conjunto de hilos de nivel usuario. | ✓ |
| En la implementación de hilos modo kernel | cuando un hilo se bloquea, no bloquea al resto de los hilos del proceso. | ✓ |
| En la implementación de hilos en modo usuario | cada hilo de nivel usuario tiene algún conjunto de hilos de nivel kernel. | ✗ |

Respuesta parcialmente correcta.

Ha seleccionado correctamente 2.

La respuesta correcta es: En la implementación de hilos de manera híbrida → cada hilo de nivel kernel tiene algún conjunto de hilos de nivel usuario., En la implementación de hilos modo kernel → cuando un hilo se bloquea, no bloquea al resto de los hilos del proceso., En la implementación de hilos en modo usuario → la creación de nuevos hilos es mas rápida.

Pregunta **10**

Parcialmente correcta

Puntúa 0,50 sobre 1,00

Un proceso posee 1+nh hilos (main + nh). Los hilos incrementan las variables total.

1- Las sumas se sincronizan con un semáforo sin nombre.

2- El hilo main espera a que el resto de los hilos terminen y muestra el valor de total.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>

int t, total, c;
sem_t sem;

void *hilo0 (void * nro) {
    int numero, j;
    numero = *(int*)nro;
    sem_unlink(&sem);
    for(j=0; j < c; j++){
        sem_wait(&sem);
        total = total + numero;
        //linea en blanco
    }
    pthread_exit(NULL);
}

int main()    {
    int nh,j,a[1];
    a[0]=10;
    nh = 10;
    pthread_t h[nh];
    total = 10000;
    c = 10000;
    sem_init(&sem, 0, 1);
    for(t=0; t < nh; t++){
        pthread_create(&h[t], NULL, hilo0, (void *) (&a[0]));
    }
    for(t=0; t < nh; t++){
        pthread_join(h[t],NULL);
    }
    sem_destroy(&sem);
    printf("total = %d\n",total);
    pthread_exit(NULL);
}

/*
En consola vemos
total= 102000
*/
```

pthread_detach(h[t],NULL);

sem_init(&sem, 1, 0);

pthread_join(hilo[t],NULL);	
1010000	
10010000	sem_init(&sem, 0, 0);
pthread_create(&a[t], NULL, hilo0, (void *) (&a[t]))	sem_post(&sem);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 3.

La respuesta correcta es:

Un proceso posee 1+nh hilos (main + nh). Los hilos incrementan las variables total.

1- Las sumas se sincronizan con un semáforo sin nombre.

2- El hilo main espera a que el resto de los hilos terminen y muestra el valor de total.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>

int t, total, c;
sem_t sem;

void *hilo0 (void * nro) {
    int numero, j;
    numero = *(int*)nro;
    [//linea en blanco]
    for(j=0; j < c; j++){
        sem_wait(&sem);
        total = total + numero;
        [sem_post(&sem);]
    }
    pthread_exit(NULL);
}

int main()    {
    int nh,j,a[1];
    a[0]=10;
    nh = 10;
    pthread_t h[nh];
    total = 10000;
    c = 10000;
    [sem_init(&sem, 0, 1);]
    for(t=0; t < nh; t++){
        [pthread_create(&h[t], NULL, hilo0, (void *) (&a[0]))];]
    }
    for(t=0; t < nh; t++){
        [pthread_join(h[t],NULL);]
    }
    sem_destroy(&sem);
    printf("total = %d\n",total);
    pthread_exit(NULL);
}

/*
En consola vemos
total= [1010000]
*/
```

Pregunta **11**

Sin contestar

Puntúa como 1,00

ENUNCIADO

Modifique el archivo padre.c para que reciba del proceso hijo 3 caracteres mediante la cola de mensajes declarada como MQ_PATH y los imprima por consola.

El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr. Se recomienda que el padre elimine la cola de mensajes luego de leer el mensaje enviado por el proceso hijo.

El proceso hijo recibe como parámetro la variable mqd. Escribe en la cola de mensaje 3 caracteres y cierra la cola de mensaje. Luego de realizar estas acciones, escribe por consola los siguientes mensajes de confirmación:

Hijo en ejecucion...

Hijo: mensaje enviado.

Hijo: cola de mensajes cerrada.

PASOS A SEGUIR

1) Descargue los siguientes 2 archivos en un mismo directorio:

1.a) Código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/5WnAqmncswdASWQ>

1.b) Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/FncdSNegcP9JTKq>

2) Complete el archivo padre.c. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr.

3) Copiar y pegar en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4) El binario generado por el comando anterior se ejecuta en consola con:

```
./padre
```

5) Luego que el proceso padre reciba los 3 caracteres los debe imprimir por consola:

```
XXX
```

RESPUESTA DEL EJERCICIO

Copie el número de 3 cifras XXX en la casilla de abajo. Este número es la respuesta del ejercicio.

Respuesta:



La respuesta correcta es: 241



Pregunta **12**

Parcialmente correcta

Puntúa 0,12 sobre 0,73

Indique lo correcto para RTOS expropiativos.

Seleccione una o más de una:

- ☐ a. La unidad básica de planificación son las tareas, y las mismas no deben bloquearse para evitar aumentar la latencia.
- ☐ b. Este tipo de RTOS permite disminuir el tiempo de latencia de foreground respecto a los RTOS no apropiativos.
- ☒ c. El uso correcto de semáforos o colas de mensaje permite evitar problemas de incoherencia. 
- ☐ d. Al seleccionar incorrectamente las prioridades de tareas foreground se puede presentar incoherencia de datos.
- ☒ e. Este tipo de planificadores hace un cambio de contexto cada vez que haya una tarea de foreground de igual o mas prioridad en estado listo (ready). 

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: El uso correcto de semáforos o colas de mensaje permite evitar problemas de incoherencia., Este tipo de RTOS permite disminuir el tiempo de latencia de foreground respecto a los RTOS no apropiativos.

[◀ Recuperatorio Parcial 1](#)

Ir a...

Comenzado el	jueves, 24 de junio de 2021, 19:00
Estado	Finalizado
Finalizado en	jueves, 24 de junio de 2021, 19:57
Tiempo empleado	56 minutos 57 segundos
Vencido	1 minutos 57 segundos
Puntos	5,08/10,00
Calificación	5,07 de 10,00 (51%)

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe enviar a través de una cola de mensajes LIFO el valor entero 123 de manera continua.
- 2- En el caso de que la cola esté llena, vTask1 se debe bloquear durante 1 segundo.
- 3- La tarea vTask2 debe recibir los datos enviados por vTask1.
- 4- En el caso de que la cola esté vacía, vTask2 se debe bloquear durante 500ms.

```
QueueHandle_t xQueue;
int main( void )
{
    UBaseType_t uxQueueLength = 10;
    UBaseType_t uxItemSize = sizeof( int32_t );
    xQueue = [input] [check] ( [input] [check] );

    /*Creación de tareas, inicio planificador*/
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    int32_t lValueToSend = 123;
    const TickType_t xDelay = [input] [check] ( 1000 );

    for ( ;; ) {
        [input] [cross] ( xQueue, [input] [cross] );
    }
}

void vTask2 ( void *pvParameters )
{
    int32_t lReceivedValue;
    const TickType_t xDelay = [input] [check] ( 500 );

    for ( ;; ) {
        [input] [check] ( xQueue, [input] [cross] );
        vPrintString( "Dato recibido\r\n" );
    }
}
```

&lValueToSend, xDelay

&lReceivedValue, xDelay

xQueue

xQueueSendToFront

uxItemSize

xQueueSendToBackFromISR

lValueToSend

pdMS_TO_TICKS

lReceivedValue

&lReceivedValue, portMAX_DELAY

&lValueToSend, portMAX_DELAY

xQueueReceive

xQueueSendToFrontFromISR

uxQueueLength, uxItemSize

xDelay

portYIELD_FROM_ISR

portMAX_DELAY

xHigherPriorityTaskWoken

uxQueueLength

&lValueToSend

xQueueCreate

xQueueSendToBack

xQueueReceiveFromISR

Respuesta parcialmente correcta.

Ha seleccionado correctamente 5.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe enviar a través de una cola de mensajes LIFO el valor entero 123 de manera continua.
- 2- En el caso de que la cola esté llena, vTask1 se debe bloquear durante 1 segundo.
- 3- La tarea vTask2 debe recibir los datos enviados por vTask1.
- 4- En el caso de que la cola esté vacía, vTask2 se debe bloquear durante 500ms.

```
QueueHandle_t xQueue;

int main( void )
{
    UBaseType_t uxQueueLength = 10;
    UBaseType_t uxItemSize = sizeof( int32_t );

    xQueue = [xQueueCreate]( [uxQueueLength, uxItemSize] );

    /*Creación de tareas, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    int32_t lValueToSend = 123;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        [xQueueSendToFront]( xQueue, [&lValueToSend, xDelay] );
    }
}

void vTask2 ( void *pvParameters )
{
    int32_t lReceivedValue;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 500 );

    for ( ;; ) {
        [xQueueReceive]( xQueue, [&lReceivedValue, portMAX_DELAY] );
        vPrintString( "Dato recibido\r\n" );
    }
}
```

Pregunta **2**

Correcta

Puntúa 0,71 sobre 0,71

Cuál de las siguientes afirmaciones es verdadera respecto al uso de hilos.

Seleccione una o más de una:

- ☐ a. Comparten el registro de estado.
- ☐ b. Nunca es necesaria la sincronización de hilos.
- ☒ c. La ejecución de hilos es separada y los recursos son compartidos.
- ☐ d. Es mas rápida la creación y destrucción de hilos respecto de subrutinas.
- ☒ e. Puede mejorar la velocidad de ejecución en sistemas multiprocesador.



Respuesta correcta

Las respuestas correctas son: La ejecución de hilos es separada y los recursos son compartidos., Puede mejorar la velocidad de ejecución en sistemas multiprocesador.

Pregunta **3**

Parcialmente correcta

Puntúa 0,24 sobre 0,71

Indique cuál de las siguientes son condiciones necesarias para que exista deadlock entre dos o más eventos.

Seleccione una o más de una:

- ☐ a. Sin expropiación de los eventos.
- ☐ b. Serialización de los eventos.
- ☒ c. Sin contención ni espera entre los eventos.
- ☒ d. Exclusión mutua de los eventos.
- ☒ e. Espera circular entre los eventos.
- ☐ f. Espera activa de los eventos.



Respuesta parcialmente correcta.

Ha seleccionado correctamente 2.

Las respuestas correctas son: Exclusión mutua de los eventos., Espera circular entre los eventos., Sin expropiación de los eventos.

Pregunta **4**

Parcialmente correcta

Puntúa 0,12 sobre 0,71

Seleccione lo verdadero respecto a algoritmos de reemplazo de página para memoria virtual.

Seleccione una o más de una:

- ☐ a. El algoritmo NRU es más fácil de implementar que el algoritmo working set.
- ☐ b. El algoritmo NRU es una mejora del algoritmo clock.
- ☒ c. El algoritmo LRU es una simplificación del NFU y se ejecuta más rápido.
- ☐ d. El algoritmo óptimo no se puede implementar genéricamente y no tiene uso.
- ☒ e. El algoritmo wsclock es una mejora del algoritmo working set.

✗

✓

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: El algoritmo NRU es más fácil de implementar que el algoritmo working set., El algoritmo wsclock es una mejora del algoritmo working set.

Pregunta **5**

Parcialmente correcta

Puntúa 0,07 sobre 0,72

Seleccione lo correcto referido a la implementación foreground/background (primer plano/segundo plano) para RTS.

Seleccione una o más de una:

- ☐ a. Pueden tener incoherencia si los datos se comparten entre rutinas de atención de interrupción y tareas de primer plano.
- ☐ b. La latencia en estos sistemas está definida por el tiempo que demora en ejecutar el bucle de foreground.
- ☒ c. La latencia en estos sistemas dependerá de lo que demore la ejecución de las rutinas de atención de interrupción.
- ☐ d. Para solucionar posibles problemas de incoherencia de datos se puede hacer uso de semáforos o mutex.
- ☐ e. La ventaja de estos sistemas respecto de bucle scan es que son mas sencillos de programar.
- ☒ f. Los sistemas foreground/background tienen mayor eficiencia que los sistemas scan loop.
- ☒ g. Pueden tener incoherencia si los datos se comparten entre tareas de primer plano.

✓

✗

✗

Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

Las respuestas correctas son: La latencia en estos sistemas dependerá de lo que demore la ejecución de las rutinas de atención de interrupción., Pueden tener incoherencia si los datos se comparten entre rutinas de atención de interrupción y tareas de primer plano.

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea vTask1 que se ejecute de manera continua e informe este estado por el puerto serie.
- 2- vTask1 se debe crear en primer lugar.
- 3- Posea una tarea vTask2 que pase a estado "Lista" exactamente cada 750ms y cambie el estado del LED verde.

```
int main( void )
{
    [ ] ✓ ( [ ] ✓ , NULL, configMINIMAL_STACK_SIZE, NULL,
[ ] ✓ , NULL );
    [ ] ✓ ( [ ] ✓ , NULL, configMINIMAL_STACK_SIZE, NULL,
[ ] ✓ , NULL );
    [ ] ✓
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for( ;; ) {
        vPrintString( "Tarea continua en ejecución\r\n" );
    }
}

void vTask2 ( void *pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = [ ] ✓ ( 750 );

    xLastWakeTime = [ ] ✓ ;

    for( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [ ] ✓ ( [ ] ✗ );
    }
}
```

void vTask1	xTaskGetTickCount()	vTaskPrioritySet	vTaskDelete(NULL)
xPeriod	vTaskStartScheduler();	for(;;)	pvParameters
tskIDLE_PRIORITY+3		xTaskCreate	vPrintString
	vTask1	750	vTaskDelayUntil
vTask2	xLastWakeTime	&xLastWakeTime, xPeriod	xTaskHandle
vTaskDelay	pdMS_TO_TICKS	xDelay	
tskIDLE_PRIORITY+2	void vTask2		

Respuesta parcialmente correcta.

Ha seleccionado correctamente 10.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea vTask1 que se ejecute de manera continua e informe este estado por el puerto serie.
- 2- vTask1 se debe crear en primer lugar.
- 3- Posea una tarea vTask2 que pase a estado "Lista" exactamente cada 750ms y cambie el estado del LED verde.

```
int main( void )
{
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+2], NULL );
    [xTaskCreate]( [vTask2], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+3], NULL );

    [vTaskStartScheduler()];
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for( ;; ) {
        vPrintString( "Tarea continua en ejecución\r\n" );
    }
}

void vTask2 ( void *pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = [pdMS_TO_TICKS]( 750 );

    xLastWakeTime = [xTaskGetTickCount()];
    for( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [vTaskDelayUntil]( [&xLastWakeTime, xPeriod] );
    }
}
```

Pregunta **7**

Parcialmente correcta

Puntúa 0,36 sobre 0,71

Indique lo correcto para el algoritmo de planificación Fair-Share.

Seleccione una o más de una:

- ☐ a. Usa técnicas de envejecimiento.
- ☐ b. Es un sistema proporcional para las tareas.
- ☐ c. El planificador tiene prioridades preestablecidas.
- ☐ d. disminuye el tiempo de retorno
- ☐ e. Es un algoritmo expropiativo.
- ☒ f. Es un sistema proporcional para los usuarios.



Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Es un algoritmo expropiativo., Es un sistema proporcional para los usuarios.

Pregunta **8**

Incorrecta

Puntúa 0,00 sobre 0,71

Seleccione lo correcto respecto algoritmos de ubicación de memoria, para gestión de memoria con particiones variables.

Seleccione una o más de una:

- ☐ a. El siguiente ajuste tiene rendimiento un poco peor que el primer ajuste.
- ☐ b. El mejor ajuste provoca más desperdicio de memoria que el primer ajuste.
- ☒ c. El primer ajuste es más lento para liberar memoria que el ajuste rápido.
- ☐ d. El primer ajuste es más lento para asignar memoria que el peor ajuste.
- ☒ e. El peor ajuste es mucho más lento para asignar memoria que el mejor ajuste.

✗

✗

Respuesta incorrecta.

Las respuestas correctas son: El siguiente ajuste tiene rendimiento un poco peor que el primer ajuste., El mejor ajuste provoca más desperdicio de memoria que el primer ajuste.

Pregunta 9

Parcialmente correcta

Puntúa 0,83 sobre 1,00

Este programa es la implementación productor-consumidor sincronizado con semáforos sin nombre.

El hilo main crea al hilo consumidor y es el hilo productor.

Completar.

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>

#define C    2
#define V    10

static int b[C] = {0};
pthread_t th1, th2;
sem_t sem_dato, sem_lugar;

void prd(void) {
    int dato, i, pos;
    pos = 0;
    dato = 100;
    for(i=0; i < V; i++ ) {
         ✓
        sem_wait(&sem_lugar);
        dato++;
        b[pos] = dato;
        pos++;
        if (pos >= C) {pos=0;}
        printf("Produce dato: %d\n", dato);
         ✓
    }
    pthread_exit(0);
}

int main() {
    int dato, i, pos;
     ✓ //inicilizar semáforo sem_lugar
     ✓ //inicilizar semáforo sem_dato
    pthread_create(&th1, NULL, (void*)&prd, NULL);
    pos = 0;
    for(i=0; i < V; i++ ) {
         ✓
        dato = b[pos];
        pos++ ;
        if (pos>= C) {pos=0;}
        printf("Consume dato: %d\n", dato);
        sem_post(&sem_lugar);
         ✗
    }
    pthread_exit(0);
}
```

//linea en blanco

sem_init(&sem_dato, 0, C);

sem_wait(&sem_dato);

sem_init(&sem_lugar, 0, 0);

sem_init(&sem_lugar, 0, C);

sem_init(&sem_dato, 0, V);

sem_post(&sem_dato);

sem_init(&sem_dato, 0, 0);

pthread_join(&th1, NULL);

sem_init(&sem_lugar, 0, V);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 5.

La respuesta correcta es:

Este programa es la implementación productor-consumidor sincronizado con semáforos sin nombre.

El hilo main crea al hilo consumidor y es el hilo productor.

Completar.

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>

#define C    2
#define V    10

static int b[C] = {0};
pthread_t th1, th2;
sem_t sem_dato, sem_lugar;

void prd(void) {
    int dato, i, pos;
    pos = 0;
    dato = 100;
    for(i=0; i < V; i++ ) {
        [//línea en blanco]
        sem_wait(&sem_lugar);
        dato++;
        b[pos] = dato;
        pos++;
        if (pos >= C) {pos=0;}
        printf("Produce dato: %d\n", dato);
        [sem_post(&sem_dato);]
    }
    pthread_exit(0);
}

int main() {
    int dato, i, pos;
    [sem_init(&sem_lugar, 0, C);] //inicilizar semáforo sem_lugar
    [sem_init(&sem_dato, 0, 0);] //inicilizar semáforo sem_dato
    pthread_create(&th1, NULL, (void*)&prd, NULL);
    pos = 0;
    for(i=0; i < V; i++ ) {
        [sem_wait(&sem_dato);]
        dato = b[pos];
        pos++ ;
        if (pos>= C) {pos=0;}
        printf("Consume dato: %d\n", dato);
        sem_post(&sem_lugar);
        [//línea en blanco]
    }
    pthread_exit(0);
}
```

Pregunta **10**

Parcialmente correcta

Puntúa 0,55 sobre 0,73

Indique lo correcto referido a RTOS no apropiativos.

Seleccione una o más de una:

- ☐ a. En este tipo de RTOS no existen los problemas de incoherencia de datos.
- ☐ b. Es más eficiente que un sistema foreground/background.
- ☒ c. Permiten gestionar mejor las tareas de background que en los sistemas foreground/background.
- ☒ d. Tienen mayor latencia las tareas de foreground que en en los RTOS apropiativos.
- ☐ e. Es más fácil la gestión de ISR en los RTOS no apropiativos que en los RTOS apropiativos.



Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

La respuesta correcta es: Tienen mayor latencia las tareas de foreground que en en los RTOS apropiativos.

Pregunta **11**

Parcialmente correcta

Puntúa 0,67 sobre 1,00

Un proceso posee 3 hilos: main, hilo0, hilo1. Los 3 hilos incrementan las variables total y total1.

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- El hilo1 debe esperar al hilo0 antes de acceder a las variables total y total1,
- 3- El hilo main debe esperar al hilo1 antes de acceder a las variables total y total1.
- 4- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;

void *hilo0 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
     ✓
    for(t=0; t < 100000 ; t++){
        total=total+numero;
        total1=total1+1;
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
     ✓
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int j, a[5]={0};
    total=10000;
    total1=100000;
    a[0]=10;
    a[1]=2;
    a[3]=1;
     ✓
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
     ✓
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+1;
    }
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total=  ✗ ,Total1=  ✗
*/
```


	//linea en blanco	pthread_exit(NULL);
	pthread_join(h[0],NULL);	320000
3200000	pthread_join(h[2],NULL);	11200000
pthread_join(h[1],NULL);	pthread_yield();	1120000
1210000	pthread_detach(pthread_self());	pthread_mutexattr_init(&mtxattr);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 4.

La respuesta correcta es:

Un proceso posee 3 hilos: main, hilo0, hilo1. Los 3 hilos incrementan las variables total y total1.

- 1- El hilo main crea los hilos hilo0 e hilo1.
- 2- El hilo1 debe esperar al hilo0 antes de acceder a las variables total y total1,
- 3- El hilo main debe esperar al hilo1 antes de acceder a las variables total y total1.
- 4- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

```

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1;

void *hilo0 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    [//linea en blanco]
    for(t=0; t < 100000 ; t++){
        total=total+numero;
        total1=total1+1;
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
    int numero, t;
    numero = *(int*)nro;
    [pthread_join(h[0],NULL);]
    for(t=0; t < 10000 ; t++){
        total=total+1;
        total1=total1+numero;
    }
    pthread_exit(NULL);
}

int main() {
    int j, a[5]={0};
    total=10000;
    total1=100000;
    a[0]=10;
    a[1]=2;
    a[3]=1;
    [//linea en blanco]
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
    [pthread_join(h[1],NULL);]
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+1;
    }
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}

/*
En consola vemos
Total= [1120000] ,Total1= [320000]
*/

```

ENUNCIADO

Modifique el archivo padre.c para que reciba del proceso hijo 3 caracteres mediante la cola de mensajes declarada como MQ_PATH y los imprima por consola.

El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr. Se recomienda que el padre elimine la cola de mensajes luego de leer el mensaje enviado por el proceso hijo.

El proceso hijo recibe como parámetro la variable mqd. Escribe en la cola de mensaje 3 caracteres y cierra la cola de mensaje. Luego de realizar estas acciones, escribe por consola los siguientes mensajes de confirmación:

Hijo en ejecucion...

Hijo: mensaje enviado.

Hijo: cola de mensajes cerrada.

PASOS A SEGUIR

1) Descargue los siguientes 2 archivos en un mismo directorio:

1.a) Código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/5WnAqmnswdASWQ>

1.b) Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/mcLnwrPaQg7PFNf>

2) Complete el archivo padre.c. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura con los parámetros indicados por la variable attr.

3) Copiar y pegar en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4) El binario generado por el comando anterior se ejecuta en consola con:

```
./padre
```

5) Luego que el proceso padre reciba los 3 caracteres los debe imprimir por consola:

```
XXX
```

RESPUESTA DEL EJERCICIO

Copie el número de 3 cifras XXX en la casilla de abajo. Este número es la respuesta del ejercicio.

Respuesta:



La respuesta correcta es: 470

[◀ Recuperatorio Parcial 1](#)

Ir a...

Comenzado el	Thursday, 2 de September de 2021, 19:01
Estado	Finalizado
Finalizado en	Thursday, 2 de September de 2021, 19:55
Tiempo empleado	54 minutos 27 segundos
Puntos	6,69/10,00
Calificación	6,69 de 10,00 (67%)

Pregunta **1**

Correcta

Puntúa 0,73 sobre 0,73

La inversión de prioridad:

- ☒ a. Se puede dar entre dos tareas de distinta prioridad que utilicen un semáforo binario compartido. ✓
- ☐ b. Intercambia las prioridades de las tareas involucradas, quedando con menos prioridad la tarea que era más prioritaria.
- ☒ c. Siempre aumenta la latencia de la tarea más prioritaria. ✓
- ☐ d. Puede ser evitada utilizando un mutex en lugar de un semáforo binario en FreeRTOS.
- ☒ e. Se puede dar entre dos tareas de distinta prioridad que utilicen una cola de mensajes compartida. ✓

Respuesta correcta

Las respuestas correctas son: Se puede dar entre dos tareas de distinta prioridad que utilicen un semáforo binario compartido., Se puede dar entre dos tareas de distinta prioridad que utilicen una cola de mensajes compartida., Siempre aumenta la latencia de la tarea más prioritaria.

Pregunta **2**

Incorrecta

Puntúa 0,00 sobre 0,71

Seleccione qué afirmación es correcta respecto algoritmos de ubicación de memoria, para gestión de memoria con particiones variables.

Seleccione una o más de una:

- ☐ a. El siguiente ajuste tiene rendimiento mejor que el mejor ajuste.
- ☐ b. El peor ajuste es mucho más lento para asignar memoria que el mejor ajuste.
- ☐ c. El algoritmo del mejor ajuste se puede mejorar con listas separadas y ordenadas de huecos y procesos.
- ☒ d. El siguiente ajuste es más lento para asignar memoria que el peor ajuste. ✗
- ☒ e. El primer ajuste es más lento para liberar memoria que el ajuste rápido. ✗

Respuesta incorrecta.

Las respuestas correctas son: El siguiente ajuste tiene rendimiento mejor que el mejor ajuste., El algoritmo del mejor ajuste se puede mejorar con listas separadas y ordenadas de huecos y procesos.

Pregunta **3**

Parcialmente correcta

Puntúa 0,89 sobre 1,00

Un proceso posee 3 hilos (main, hilo1, hilo2).

- 1- El hilo main crea los hilos hilo1 e hilo2.
- 2- Los hilos hilo1 e hilo2 incrementan las variables total y total1.
- 3- El hilo main incrementa la variable total.
- 4- El hilo main espera a que los hilos terminen y muestra el resultado de total y total1.

Completar

```

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1, a[5]={0};
pthread_mutex_t b0=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t b1=PTHREAD_MUTEX_INITIALIZER;

void *hilo1 (void * nro) {
int d,t;
d = *(int*)nro;
pthread_mutex_lock(&b0); ✓
pthread_mutex_lock(&b1); ✓
for(t=0; t < 100000 ; t++){
total=total+d;
total1=total1+a[0];
//línea en blanco ✓
}
pthread_mutex_unlock(&b1);
pthread_mutex_unlock(&b0);
pthread_exit(NULL);
}

void *hilo2 (void * nro) {
int d,t;
d = *(int*)nro;
pthread_mutex_lock(&b0);
pthread_mutex_lock(&b1);
for(t=0; t < 10000 ; t++){
total=total+a[1];
total1=total1+a[2];
//línea en blanco ✓
}
pthread_mutex_unlock(&b1); ✓
pthread_mutex_unlock(&b0);
pthread_exit(NULL);
}

int main() {
int t;
a[0]=1;
a[1]=2;
a[3]=2;
total=20000;
total1=1000;
pthread_create(&h[0], NULL, hilo1, (void *) (&a[1]) );
pthread_create(&h[1], NULL, hilo2, (void *) (&a[1]) );
pthread_mutex_lock(&b0); ✓
for(t=0; t < 1000000 ; t++){
//línea en blanco ✓
total=total+a[0];
total=total+a[3];
}

pthread_mutex_unlock(&b0);
pthread_join(h[0],NULL);
pthread_join(h[1],NULL);
printf("Total= %d, Total1= %d\n",total,total1);
pthread_exit(NULL);
}
/*
En consola vemos
Total= 102000 ✗ , Total1= 101000 ✓
*/

```

102000	pthread_mutex_lock(&b1);	324000	2300000
101000	pthread_join(h[0],NULL);	pthread_mutex_unlock(&b1);	pthread_mutex_unlock(&b0);
pthread_join(h[1],NULL);	3231000	//linea en blanco	pthread_mutex_lock(&b0);
111000	3240000		

Respuesta parcialmente correcta.

Ha seleccionado correctamente 8.

La respuesta correcta es:

Un proceso posee 3 hilos (main, hilo1, hilo2).

- 1- El hilo main crea los hilos hilo1 e hilo2.
- 2- Los hilos hilo1 e hilo2 incrementan las variables total y total1.
- 3- El hilo main incrementa la variable total.
- 4- El hilo main espera a que los hilos terminen y muestra el resultado de total y total1.

Completar

```

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

pthread_t h[2];
int total, total1, a[5]={0};
pthread_mutex_t b0=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t b1=PTHREAD_MUTEX_INITIALIZER;

void *hilo1 (void * nro) {
int d,t;
    d = *(int*)nro;
    [pthread_mutex_lock(&b0);]
    [pthread_mutex_lock(&b1);]
    for(t=0; t < 100000 ; t++){
        total=total+d;
        total1=total1+a[0];
        [//linea en blanco]
    }
    pthread_mutex_unlock(&b1);
    pthread_mutex_unlock(&b0);
    pthread_exit(NULL);
}

void *hilo2 (void * nro) {
int d,t;
    d = *(int*)nro;
    pthread_mutex_lock(&b0);
    pthread_mutex_lock(&b1);
    for(t=0; t < 10000 ; t++){
        total=total+a[1];
        total1=total1+a[2];
        [//linea en blanco]
    }
    [pthread_mutex_unlock(&b1);]
    pthread_mutex_unlock(&b0);
    pthread_exit(NULL);
}

int main() {
int t;
    a[0]=1;
    a[1]=2;
    a[3]=2;
    total=20000;
    total1=1000;
    pthread_create(&h[0], NULL, hilo1, (void *) (&a[1]) );
    pthread_create(&h[1], NULL, hilo2, (void *) (&a[1]) );
    [pthread_mutex_lock(&b0);]
    for(t=0; t < 1000000 ; t++){
        [//linea en blanco]
        total=total+a[0];
        total=total+a[3];
    }

    pthread_mutex_unlock(&b0);
    pthread_join(h[0],NULL);
    pthread_join(h[1],NULL);
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}
/*
En consola vemos
Total=[3240000] ,Total1= [101000]
*/

```


Pregunta 4

Correcta

Puntúa 1,00 sobre 1,00

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Utilizando un mecanismo de sincronización, vTask1 debe cambiar el estado del LED verde cuando se recibe una interrupción del RIT.
- 2- vTask1 debe esperar la señal de sincronización durante 1 segundo.
- 3- vTask1 es de alta prioridad. vTask 2 ejecuta una espera activa de baja prioridad.

```

SemaphoreHandle_t xBinarySemaphore;
SemaphoreHandle_t xMutex;
int main( void )
{ ... }

void vTask1 ( void *pvParameters )
{
    const TickType_t xDelay = pdMS_TO_TICKS ( 1000 );

    for ( ;; ) {
        if( xSemaphoreTake ( xBinarySemaphore, xDelay ) ==
pdTRUE) Board_LED_Toggle(5); //LED verde
        else Board_LED_Toggle(4); //LED rojo
    }
}

void vTask2 ( void *pvParameters )
{
    for ( ;; ) { vPrintString( "Espera activa\r\n" ); }
}

void RIT_IRQHandler(void)
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    xSemaphoreGiveFromISR ( xBinarySemaphore , &xHigherPriorityTaskWoken );
    if( xHigherPriorityTaskWoken == pdTRUE ) {
        portYIELD_FROM_ISR ( xHigherPriorityTaskWoken );
    }
    Chip_RIT_ClearInt(LPC_RITIMER);
}

```

xBinarySemaphore, portMAX_DELAY	xSemaphoreCreateBinary()	portYIELD_FROM_ISR
xSemaphoreCreateMutex()	xDelay	xSemaphoreGiveFromISR
xBinarySemaphore, xDelay	xSemaphoreTakeFromISR	xSemaphoreGive
xSemaphoreTake	xMutex, xDelay	xBinarySemaphore
pdMS_TO_TICKS	xMutex, portMAX_DELAY	xMutex
xHigherPriorityTaskWoken		

Respuesta correcta

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Utilizando un mecanismo de sincronización, vTask1 debe cambiar el estado del LED verde cuando se recibe una interrupción del RIT.
- 2- vTask1 debe esperar la señal de sincronización durante 1 segundo.
- 3- vTask1 es de alta prioridad. vTask 2 ejecuta una espera activa de baja prioridad.

```
SemaphoreHandle_t xBinarySemaphore;
SemaphoreHandle_t xMutex;
int main( void )
{ ... }

void vTask1 ( void *pvParameters )
{
const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        if( [xSemaphoreTake]( [xBinarySemaphore, xDelay] ) == pdTRUE) Board_LED_Toggle(5); //LED verde
        else Board_LED_Toggle(4); //LED rojo
    }
}

void vTask2 ( void *pvParameters )
{
    for ( ;; ) { vPrintString( "Espera activa\r\n" ); }
}

void RIT_IRQHandler(void)
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    [xSemaphoreGiveFromISR]( [xBinarySemaphore], &xHigherPriorityTaskWoken );
    if( [xHigherPriorityTaskWoken] == pdTRUE ) {
        [portYIELD_FROM_ISR]( [xHigherPriorityTaskWoken] );
    }
    Chip_RIT_ClearInt(LPC_RITIMER);
}
```

Pregunta **5**

Incorrecta

Puntúa 0,00 sobre 0,71

Indique cuál de los siguientes es un recurso compartido por los hilos de un mismo proceso.

Seleccione una o más de una:

- ☒ a. Espacio de direcciones (space address).
- ☐ b. Memoria dinámica (heap).
- ☒ c. Registros de estado.
- ☐ d. Memoria de programa (text).
- ☐ e. Pila (stack).



Respuesta incorrecta.

El objetivo es que comprenda los recursos usados por los hilos de un proceso.

Las respuestas correctas son: Espacio de direcciones (space address)., Memoria dinámica (heap)., Memoria de programa (text).

Pregunta **6**

Correcta

Puntúa 0,71 sobre 0,71

Seleccione qué afirmación es correcta respecto a segmentación para memoria virtual.

Seleccione una o más de una:

- ☒ a. Cada segmento constituye un espacio de direcciones independiente. ✓
- ☒ b. Desacopla la relación 1 a 1 entre el bus de direcciones y la memoria física. ✓
- ☒ c. Para acceder a una dirección física solo es necesario indicar el nombre del segmento y el offset. ✓
- ☐ d. La memoria virtual se divide en segmentos independientes de igual tamaño y potencia de 2.
- ☐ e. El objetivo principal es ejecutar programas tan grandes que la memoria física.

Respuesta correcta

Las respuestas correctas son: Cada segmento constituye un espacio de direcciones independiente., Desacopla la relación 1 a 1 entre el bus de direcciones y la memoria física., Para acceder a una dirección física solo es necesario indicar el nombre del segmento y el offset.

Pregunta 7

Correcta

Puntúa 1,00 sobre 1,00

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea única con la prioridad más alta posible.
- 2- La tarea debe cambiar el estado del LED rojo, enviar por puerto serie el texto enviado como parámetro en su creación y bloquearse durante 800ms.

```
const char *pvTaskParameters = "Tarea 1 en ejecución\r\n";

int main( void )
{
    xTaskCreate ( vTask1 , NULL, configMINIMAL_STACK_SIZE,
(void*)pvTaskParameters, tskIDLE_PRIORITY+3 , NULL );

    vTaskStartScheduler();

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    char *pcTaskName;
    const TickType_t xDelay = pdMS_TO_TICKS ( 800 );
    pcTaskName = ( char * ) pvParameters ;

    for( ;; ) {
        vPrintString( pcTaskName )
        Board_LED_Toggle(4); //LED rojo;
        vTaskDelay ( xDelay );
    }
}
```

xDelay	vTaskDelete(NULL)	xTaskGetTickCount()	tskIDLE_PRIORITY+3
xLastWakeTime	pvParameters	vTask2	vPrintString
tskIDLE_PRIORITY+1	xPeriod	for(;;)	pdMS_TO_TICKS
&xLastWakeTime, xPeriod	vTask1	void vTask2	xTaskHandle
vTaskDelayUntil	void vTask1	vTaskStartScheduler();	vTaskPrioritySet
xTaskCreate	vTaskDelay	tskIDLE_PRIORITY+2	

Respuesta correcta

La respuesta correcta es:
Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Posea una tarea única con la prioridad más alta posible.
- 2- La tarea debe cambiar el estado del LED rojo, enviar por puerto serie el texto enviado como parámetro en su creación y bloquearse durante 800ms.

```
const char *pvTaskParameters = "Tarea 1 en ejecución\r\n";

int main( void )
{
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, (void*)pvTaskParameters, [tskIDLE_PRIORITY+3], NULL );
```

```
    [vTaskStartScheduler();]  
    for( ;; );  
    return 0;  
}  
  
void vTask1 ( void *pvParameters )  
{  
    char *pcTaskName;  
    const TickType_t xDelay = [pdMS_TO_TICKS]( 800 );  
    pcTaskName = ( char * ) [pvParameters];  
    [for( ;; )] {  
        vPrintString( pcTaskName )  
        Board_LED_Toggle(4); //LED rojo;  
        [vTaskDelay]( [xDelay] );  
    }  
}
```

Pregunta **8**

Incorrecta

Puntúa 0,00 sobre 1,00

ENUNCIADO

Modifique el archivo padre.c para que envíe al proceso hijo la cadena de caracteres "HOLA_HIJO" mediante la cola de mensajes declarada como MQ_PATH.

Por el lado del padre:

1. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura.
2. El padre debe enviar al proceso hijo la cadena de caracteres "HOLA_HIJO" mediante la cola de mensajes declarada como MQ_PATH.
3. Se recomienda que el padre elimine la cola de mensajes luego de escribir el mensaje en la cola.

Por el lado del hijo:

1. El proceso hijo recibe como parámetro la variable mqd.
2. Lee de la cola de mensajes lo enviado por el proceso padre y cierra la cola de mensaje.
3. Solo si recibe "HOLA_HIJO" escribe por consola el resultado del ejercicio.

PASOS A SEGUIR

1. Descargue los siguientes 2 archivos en un mismo directorio:

- 1.a. Plantilla del código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/A2YXLEn4krGdTwB>

- 1.b. Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/CpFBcpqnwjGcGNrM>

2. Complete el archivo padre.c según el enunciado del ejercicio.

3. Copie y pegue en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4. El binario generado por el comando anterior se ejecuta en consola con:

```
./padre
```

5. El proceso hijo imprime por consola:

```
Hijo en ejecucion...
```

```
Hijo: mensaje leído es HOLA_HIJO
```

```
Hijo: el resultado del ejercicio es *** XXX ***
```

```
Hijo: cola de mensajes cerrada.
```

6. Si el proceso hijo no recibe "HOLA_HIJO", imprime por consola:

```
Hijo: mensaje leído no coincide con HOLA_HIJO
```

RESPUESTA DEL EJERCICIO

La respuesta de este ejercicio es el número de 3 cifras XXX que imprime el proceso hijo por consola. Copie este número de 3 cifras XXX en la casilla de abajo.

Respuesta:



La respuesta correcta es: 502

Pregunta 9

Parcialmente correcta

Puntúa 0,54 sobre 0,71

Indique qué afirmación es correcta respecto a la planificación de procesos.

- ☐ a. Al crearse hilos en espacio usuario, se deben modificar los planificadores para incluirlos.
- ☐ b. Cuanto mayor sea el quantum que utilizan los planificadores, más interactivo será el sistema operativo.
- ☒ c. Con un planificador no expropiativo se ven beneficiadas las tareas I/O bounded. ✗
- ☒ d. Al poder tener en memoria más de un proceso a la vez, es necesario el uso de planificadores. ✓
- ☐ e. Los planificadores expropiativos solo se ejecutan cuando el proceso en estado READY ejecuta una syscall.

Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

La respuesta correcta es: Al poder tener en memoria más de un proceso a la vez, es necesario el uso de planificadores.

Pregunta 10

Correcta

Puntúa 0,72 sobre 0,72

Una lo que corresponda.

Un RTOS no expropiativo	<div>puede tener incoherencia solo si se comparten datos entre ISR y tareas de primer plano.</div> <div>✓</div>
Un sistema background/foreground respecto de un scan loop	<div>disminuye la latencia de las tareas de segundo plano.</div> <div>✓</div>
un RTOS expropiativo respecto a un RTOS no expropiativo	<div>disminuye la latencia de las tareas de primer plano.</div> <div>✓</div>
Un sistema scan loop	<div>tiene el mayor de los rendimientos.</div> <div>✓</div>

Respuesta correcta

La respuesta correcta es: Un RTOS no expropiativo → puede tener incoherencia solo si se comparten datos entre ISR y tareas de primer plano., Un sistema background/foreground respecto de un scan loop → disminuye la latencia de las tareas de segundo plano., un RTOS expropiativo respecto a un RTOS no expropiativo → disminuye la latencia de las tareas de primer plano., Un sistema scan loop → tiene el mayor de los rendimientos.

Pregunta **11**

Parcialmente correcta

Puntúa 0,40 sobre 1,00

Dos procesos relacionados se comunican por medio de una cola de mensajes.

- 1- El proceso padre crea la cola de mensajes.
- 2- El proceso padre crea al proceso hijo.
- 4- El proceso padre escribe un mensaje en la cola de mensajes.
- 5- El proceso padre espera a que el proceso hijo termine.
- 6- El proceso padre cierra y elimina la cola de mensajes y luego terminar él.
- 7- El proceso hijo lee un mensaje de la cola de mensajes, muestra lo leído.
- 8- El proceso hijo elimina la cola de mensajes y termina.

Completar

```
#include <mqueue.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>
```

```
#define MJ "Recuperatorio TD III"
#define SHM "/TD3"
```

```
char b[20];
mqd_t cc;
struct mq_attr mqa;
```

```
int main() {
    mqa.mq_msgsize = sizeof(b);
    mqa.mq_maxmsg = 3;
    //linea en blanco ✖
    if (fork() == 0) {
        mq_getattr(cc, &mqa);
        mq_receive(cc, b, mqa.mq_msgsize, 0); ✔
        printf("%s\n", b);
        mq_close(cc);
        exit(0);
    }else{
        cc = mq_open(SHM, O_RDWR | O_CREAT, 0777, &mqa); ✖
        mq_send(cc, MJ, strlen(MJ), 1);
        wait(NULL); ✔
        mq_close(cc);
        //linea en blanco ✖
    }
    exit(0);
}
```

mq_close(cc);

mq_send(cc, b, strlen(b), 1);

cc = mq_open(SHMM, O_RDWR, 0);

wait(NULL);

cc = mq_open(SHMM, O_WRONLY | O_CREAT, 0777, &mqa);

mq_receive(cc, b, strlen(b), 1);

mq_receive(cc, b, mqa.mq_msgsize, 0);	mq_unlink(SHM);
//línea en blanco	cc = mq_open(SHM, O_RDWR O_CREAT, 0777, &mqa);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 2.

La respuesta correcta es:

Dos procesos relacionados se comunican por medio de una cola de mensajes.

- 1- El proceso padre crea la cola de mensajes.
- 2- El proceso padre crea al proceso hijo.
- 4- El proceso padre escribe un mensaje en la cola de mensajes.
- 5- El proceso padre espera a que el proceso hijo termine.
- 6- El proceso padre cierra y elimina la cola de mensajes y luego terminar él.
- 7- El proceso hijo lee un mensaje de la cola de mensajes, muestra lo leído.
- 8- El proceso hijo elimina la cola de mensajes y termina.

Completar

```
#include <mqueue.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>

#define MJ "Recuperatorio TD III"
#define SHM "/TD3"

char b[20];
mqd_t cc;
struct mq_attr mqa;

int main() {
    mqa.mq_msgsize = sizeof(b);
    mqa.mq_maxmsg = 3;
    [cc = mq_open(SHM, O_RDWR | O_CREAT, 0777, &mqa);]
    if (fork() == 0) {
        mq_getattr(cc, &mqa);
        [mq_receive(cc, b, mqa.mq_msgsize, 0);]
        printf("%s\n", b);
        mq_close(cc);
        exit(0);
    }else{
        [//línea en blanco]
        mq_send(cc, MJ, strlen(MJ), 1);
        [wait(NULL);]
        mq_close(cc);
        [mq_unlink(SHM);]
    }
    exit(0);
}
```



Pregunta **12**

Correcta

Puntúa 0,71 sobre 0,71

Indique las afirmaciones correctas respecto a exclusión mutua del evento A respecto del evento B.

Seleccione una o más de una:

- ☐ a. El evento A debe esperar la ocurrencia del evento B, e inversamente.
- ☐ b. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un mutex para cada uno, ambos inicialmente en 1.
- ☐ c. El evento A y el evento B deben ejecutarse de manera simultánea.
- ☐ d. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un mutex, inicialmente en 0.
- ☒ e. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un semáforo, inicialmente en 1. 
- ☐ f. Para forzar la situación de exclusión mutua, la única alternativa es usar un mutex inicializado en 1.
- ☐ g. Para forzar la situación de exclusión mutua (para dos eventos) se puede usar barrier, inicialmente en 2.
- ☒ h. El evento A no debe ocurrir en el mismo momento que el evento B. 

Respuesta correcta

Las respuestas correctas son: Para forzar la situación de exclusión mutua (para dos eventos) se puede usar un semáforo, inicialmente en 1., El evento A no debe ocurrir en el mismo momento que el evento B.

[◀ Parcial 2](#)

Ir a...

[Área personal](#) / [Mis cursos](#) / [Técnicas Digitales III \(Práctica y Exámenes\) 2021](#) / [Parcial 2](#) / [Recuperatorio 2](#)

Comenzado el	Thursday, 2 de September de 2021, 19:01
---------------------	---

Estado	Finalizado
---------------	------------

Finalizado en	Thursday, 2 de September de 2021, 19:54
----------------------	---

Tiempo empleado	53 minutos 10 segundos
------------------------	------------------------

Puntos	6,13/10,00
---------------	------------

Calificación	6,13 de 10,00 (61%)
---------------------	-------------------------------------

Pregunta **1**Parcialmente
correctaPuntúa 0,18
sobre 0,73

Indique qué es correcto respecto a la tarea A que se está ejecutando en FreeRTOS.

- ☒ a. Si la tarea B que es de igual prioridad que A se pasa a estado listo, se debe esperar el quantum o que A se bloquee para realizar la conmutación y ejecutar la tarea B. ✔
- ☐ b. Si la tarea B que es de menor prioridad que A se pasa a estado listo, no es necesario esperar el quantum para que se ejecute la tarea B.
- ☐ c. Si la tarea B que es más prioritaria que A se pasa a estado listo, no es necesario esperar el quantum para realizar la conmutación y ejecutar la tarea B.
- ☒ d. Si la tarea B que es más prioritaria que A se pasa a estado listo, se debe esperar el quantum para realizar la conmutación y ejecutar la tarea B. ✘
- ☐ e. Si la tarea B que es de igual prioridad que A se pasa a estado listo, se debe esperar el quantum y que A se bloquee para realizar la conmutación y ejecutar la tarea B.
- ☐ f. Si la tarea B que es menos prioritaria que A se pasa a estado listo, se debe esperar el quantum para realizar la conmutación y ejecutar la tarea B.

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Si la tarea B que es más prioritaria que A se pasa a estado listo, no es necesario esperar el quantum para realizar la conmutación y ejecutar la tarea B., Si la tarea B que es de igual prioridad que A se pasa a estado listo, se debe esperar el quantum o que A se bloquee para realizar la conmutación y ejecutar la tarea B.

Pregunta **2**

Correcta

Puntúa 0,71
sobre 0,71

Indique qué afirmación es correcta para el algoritmo de planificación Round-Robin.

Seleccione una o más de una:

- ☐ a. Usa técnicas de envejecimiento.
- ☒ b. Si a una tarea en ejecución se le termina el quantum, va al final de la FIFO. ✓
- ☐ c. Disminuye el tiempo de retorno.
- ☒ d. Es un algoritmo equitativo. ✓
- ☐ e. Es un algoritmo no expropiativo.
- ☒ f. No es necesario que las tareas esten disponibles en el momento inicial. ✓

Respuesta correcta

Las respuestas correctas son: No es necesario que las tareas esten disponibles en el momento inicial., Si a una tarea en ejecución se le termina el quantum, va al final de la FIFO., Es un algoritmo equitativo.

Pregunta **3**

Correcta

Puntúa 0,71
sobre 0,71

Seleccione qué afirmación es correcta respecto algoritmos de ubicación de memoria, para gestión de memoria con particiones variables.

Seleccione una o más de una:

- ☒ a. El primer ajuste es más rápido para asignar memoria que el peor ajuste.
- ☐ b. El mejor ajuste tiene rendimiento mejor que el siguiente ajuste.
- ☒ c. El mejor ajuste provoca más desperdicio de memoria que el primer ajuste.
- ☐ d. El primer ajuste tiene rendimiento algo peor que el siguiente ajuste.
- ☐ e. El peor ajuste es más rápido para asignar memoria que el siguiente ajuste.



Respuesta correcta

Las respuestas correctas son: El mejor ajuste provoca más desperdicio de memoria que el primer ajuste., El primer ajuste es más rápido para asignar memoria que el peor ajuste.

Pregunta **4**

Correcta

Puntúa 0,71
sobre 0,71

De las siguientes afirmaciones, seleccione la o las que crea que son verdaderas respecto al uso de hilos.

Seleccione una o más de una:

- ☐ a. Todas las afirmaciones solo son válidas para sistemas multiprocesador o multinúcleo.
- ☐ b. La creación y destrucción de hilos es más rápida que la de subrutinas.
- ☐ c. Ante un error en tiempo de ejecución en el código del hilo, solo éste termina y su proceso asociado se sigue ejecutando.
- ☒ d. Ante un error en tiempo de ejecución en el código del hilo, se termina el proceso al que pertenece el hilo.
- ☒ e. Permiten el paralelismo real en computadoras de mas de un núcleo.



Respuesta correcta

Las respuestas correctas son: Ante un error en tiempo de ejecución en el código del hilo, se termina el proceso al que pertenece el hilo., Permiten el paralelismo real en computadoras de mas de un núcleo.

Pregunta **5**

Incorrecta

Puntúa 0,00
sobre 0,72

Indique las afirmaciones correctas respecto a RTS (Real Time Systems).

Seleccione una o más de una:

- ☒ a. El uso de un RTOS permite asegurar el determinismo para un RTS. ✗
- ☒ b. El determinismo en RTS es poder cumplir de manera predecible requisitos temporales. ✓
- ☐ c. Un RTS es aquel en el que la corrección del resultado depende del instante en que se produce o de su validez lógica.
- ☐ d. Los RTS no tienen que ejecutarse rápidamente.
- ☒ e. El periodo de muestreo (T_s) se refiere al periodo necesario para la adquisición de datos. ✗

Respuesta incorrecta.

Las respuestas correctas son: El determinismo en RTS es poder cumplir de manera predecible requisitos temporales., Los RTS no tienen que ejecutarse rápidamente.

Pregunta **6**

Correcta

Puntúa 1,00
sobre 1,00

Dos procesos relacionados se comunican por medio de una cola de mensajes.

- 1- El proceso padre crea la cola de mensajes.
- 2- El proceso padre crea al proceso hijo.
- 4- El proceso padre escribe un mensaje en la cola de mensajes.
- 5- El proceso padre espera a que el proceso hijo termine.
- 6- El proceso padre cierra y elimina la cola de mensajes y luego terminar él.
- 7- El proceso hijo lee un mensaje de la cola de mensajes, muestra lo leído.
- 8- El proceso hijo elimina la cola de mensajes y termina.

Completar

```
#include <mqueue.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>

#define MJ "Recuperatorio TD III"
#define SHM "/TD3"

char b[20];
mqd_t cc;
struct mq_attr mqa;

int main() {
    mqa.mq_msgsize = sizeof(b);
    mqa.mq_maxmsg = 3;
    cc = mq_open(SHM, O_RDWR | O_CREAT, 0777, &mqa);
    if (fork() == 0) {
        mq_getattr(cc, &mqa);
        mq_receive(cc, b, mqa.mq_msgsize, 0);
        printf("%s\n", b);
    }
```



```
mq_close(cc);
exit(0);
}else{
    //linea en blanco
mq_send(cc, MJ, strlen(MJ), 1);
    wait(NULL);
mq_close(cc);
    mq_unlink(SHM);
}
exit(0);
}
```

wait(NULL);	cc = mq_open(SHM, O_RDWR O_CREAT, 0777, &mqa);
cc = mq_open(SHMM, O_WRONLY O_CREAT, 0777, &mqa);	mq_receive(cc, b, mqa.mq_msgsize, 0);
mq_unlink(SHM);	//linea en blanco
cc = mq_open(SHMM, O_RDWR, 0);	mq_close(cc);
mq_send(cc, b, strlen(b), 1);	mq_receive(cc, b, strlen(b), 1);

Respuesta correcta

La respuesta correcta es:

Dos procesos relacionados se comunican por medio de una cola de mensajes.

- 1- El proceso padre crea la cola de mensajes.
- 2- El proceso padre crea al proceso hijo.
- 4- El proceso padre escribe un mensaje en la cola de mensajes.
- 5- El proceso padre espera a que el proceso hijo termine.
- 6- El proceso padre cierra y elimina la cola de mensajes y luego terminar él.
- 7- El proceso hijo lee un mensaje de la cola de mensajes, muestra lo leído.
- 8- El proceso hijo elimina la cola de mensajes y termina.

Completar

```
#include <mqueue.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>

#define MJ "Recuperatorio TD III"
#define SHM "/TD3"

char b[20];
mqd_t cc;
struct mq_attr mqa;

int main() {
    mqa.mq_msgsize = sizeof(b);
    mqa.mq_maxmsg = 3;
    [cc = mq_open(SHM, O_RDWR | O_CREAT, 0777, &mqa);]
    if (fork() == 0) {
        mq_getattr(cc, &mqa);
        [mq_receive(cc, b, mqa.mq_msgsize, 0);]
        printf("%s\n", b);
        mq_close(cc);
        exit(0);
    }else{
        [//linea en blanco]
        mq_send(cc, MJ, strlen(MJ), 1);
        [wait(NULL);]
        mq_close(cc);
        [mq_unlink(SHM);]
    }
    exit(0);
}
```

}

Pregunta **7**

Incorrecta

Puntúa 0,00
sobre 0,71

Seleccione qué afirmación es correcta respecto a la técnica de paginación para memoria virtual.

Seleccione una o más de una:

- ☒ a. Permite ejecutar programas más grandes que la memoria física. ✓
- ☒ b. Se divide la memoria física en páginas de igual tamaño, que es potencia de 2. ✗
- ☒ c. El bus de direcciones del procesador no está directamente conectado a la memoria física. ✓
- ☒ d. Al producirse un fallo de página siempre se debe ejecutar un algoritmo de reemplazo de páginas. ✗
- ☒ e. Se crea un segmento de código, datos y pila para cada programa en ejecución. ✗

Respuesta incorrecta.

Las respuestas correctas son: Permite ejecutar programas más grandes que la memoria física., El bus de direcciones del procesador no está directamente conectado a la memoria física.

Pregunta **8**

Incorrecta

Puntúa 0,00
sobre 1,00**ENUNCIADO**

Modifique el archivo padre.c para que envíe al proceso hijo la cadena de caracteres "HOLA_HIJO" mediante la cola de mensajes declarada como MQ_PATH.

Por el lado del padre:

1. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura.
2. El padre debe enviar al proceso hijo la cadena de caracteres "HOLA_HIJO" mediante la cola de mensajes declarada como MQ_PATH.
3. Se recomienda que el padre elimine la cola de mensajes luego de escribir el mensaje en la cola.

Por el lado del hijo:

1. El proceso hijo recibe como parámetro la variable mqd.
2. Lee de la cola de mensajes lo enviado por el proceso padre y cierra la cola de mensaje.
3. Solo si recibe "HOLA_HIJO" escribe por consola el resultado del ejercicio.

PASOS A SEGUIR

1. Descargue los siguientes 2 archivos en un mismo directorio:

- 1.a. Plantilla del código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/A2YXLEn4krGdTwB>

- 1.b. Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/QeyWd84EctXywbo>

2. Complete el archivo padre.c según el enunciado del ejercicio.
3. Copie y pegue en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4. El binario generado por el comando anterior se ejecuta en consola con:

./padre

5. El proceso hijo imprime por consola:

Hijo en ejecucion...

Hijo: mensaje leído es HOLA_HIJO

Hijo: el resultado del ejercicio es *** XXX ***

Hijo: cola de mensajes cerrada.

6. Si el proceso hijo no recibe "HOLA_HIJO", imprime por consola:

Hijo: mensaje leído no coincide con HOLA_HIJO

RESPUESTA DEL EJERCICIO

La respuesta de este ejercicio es el número de 3 cifras XXX que imprime el proceso hijo por consola. Copie este número de 3 cifras XXX en la casilla de abajo.

Respuesta:



La respuesta correcta es: 743

Pregunta **9**Parcialmente
correctaPuntúa 0,82
sobre 1,00

Este programa es la implementación productor-consumidor sincronizado con semáforos sin nombre.

El hilo main crea al hilo consumidor y es el hilo productor.

Completar.

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>

#define C    2
#define V    10
static int b[C] = {0};
pthread_t th1, th2;
sem_t sem_dato, sem_lugar;

void csd(void) {
int dato, i, pos;
    pos = 0;
    for(i=0; i < V; i++ ) {
        //linea en blanco ✓
        sem_wait(&sem_lugar ✗ );
        dato = b[pos];
        pos++ ;
        if (pos>= C) {pos=0;}
        printf("Consume dato: %d\n", dato);
        sem_post(&sem_lugar ✓ )
    }
    pthread_exit(0);
}

int main() {
int dato, i, pos;
    sem_init(&sem_lugar ✓ , 0, C ✓ ); //inicilizar semáforo sem_lugar
    sem_init(&sem_dato ✓ , 0, 0 ✓ ); //inicilizar semáforo sem_dato
    //linea en blanco ✓
    pthread_create(&th1, NULL, (void*)&csd, NULL);
    pos = 0;
    dato = 100;
    for(i=0; i < V; i++ ) {
        //linea en blanco ✓

```

```
sem_wait(&sem_dato ✖ );  
dato++;  
b[pos] = dato;  
pos++;  
if (pos >= C) {pos=0;}  
printf("Produce dato: %d\n", dato);  
sem_post(&sem_dato ✔ );  
}  
pthread_exit(0);  
}
```

//línea en blanco	sem_post(&sem_dato);	10	sem_lugar	sem_wait(&sem_dato);
sem_post(&sem_lugar);	sem_wait(&sem_lugar);	sem_dato	0	5
1	C	V		

Respuesta parcialmente correcta.

Ha seleccionado correctamente 9.

La respuesta correcta es:

Este programa es la implementación productor-consumidor sincronizado con semáforos sin nombre.

El hilo main crea al hilo consumidor y es el hilo productor.

Completar.


```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <fcntl.h>

#define C    2
#define V    10
static int b[C] = {0};
pthread_t th1, th2;
sem_t sem_dato, sem_lugar;

void csd(void) {
int dato, i, pos;
    pos = 0;
    for(i=0; i < V; i++ ) {
        //línea en blanco
        sem_wait(&[sem_dato]);
        dato = b[pos];
        pos++ ;
        if (pos>= C) {pos=0;}
        printf("Consume dato: %d\n", dato);
        sem_post(&[sem_lugar])
    }
    pthread_exit(0);    }

int main() {
int dato, i, pos;
    sem_init(&[sem_lugar], 0,[C]); //inicilizar semáforo sem_lugar
    sem_init(&[sem_dato], 0,[0]); //inicilizar semáforo sem_dato
    //línea en blanco
    pthread_create(&th1, NULL, (void*)&csd, NULL);
    pos = 0;
    dato = 100;
    for(i=0; i < V; i++ ) {
        //línea en blanco
        sem_wait(&[sem_lugar]);
        dato++;
        b[pos] = dato;
        pos++;
    }
```

```
        if (pos >= C) {pos=0;}
        printf("Produce dato: %d\n", dato);
        sem_post(&[sem_dato]);
    }
    pthread_exit(0);
}
```

Pregunta **10**

Correcta

Puntúa 1,00
sobre 1,00

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

1- La tarea vTask1 debe ejecutarse una única vez y debe ser la primera en ejecutarse.

2- vTask2 debe ser la primer tarea en crearse.

3- vTask2 debe pasar a estado "Lista" exactamente cada 300ms y cambiar el estado del LED verde.

```
int main( void )
{
    xTaskCreate ( vTask2 , NULL, configMINIMAL_STACK_SIZE, NULL,
    tskIDLE_PRIORITY+1 , NULL );
    xTaskCreate ( vTask1 , NULL, configMINIMAL_STACK_SIZE, NULL,
    tskIDLE_PRIORITY+2 , NULL );
    vTaskStartScheduler() ;
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for ( ;; ) {
        vPrintString( "Ejecución única\r\n" );
        vTaskDelete( NULL ) ;
    }
}

void vTask2 ( void *pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = pdMS_TO_TICKS ( 300 );
    xLastWakeTime = xTaskGetTickCount() ;
    for ( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        vTaskDelayUntil ( &xLastWakeTime, xPeriod );
    }
}
```

}

for (;;)	300	xLastWakeTime	vTaskDelay	pdMS_TO_TICKS
void vTask2	xTaskHandle	xTaskGetTickCount()	pvParameters	vTask1
vTaskDelete(NULL)	tskIDLE_PRIORITY+2	vTaskPrioritySet	vTaskDelayUntil	vTask2
tskIDLE_PRIORITY+1	vPrintString	vTaskStartScheduler()	xPeriod	void vTask1
xDelay	&xLastWakeTime, xPeriod		xTaskCreate	

Respuesta correcta

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe ejecutarse una única vez y debe ser la primera en ejecutarse.
- 2- vTask2 debe ser la primer tarea en crearse.
- 3- vTask2 debe pasar a estado "Lista" exactamente cada 300ms y cambiar el estado del LED verde.

```
int main( void )
{
    [xTaskCreate]( [vTask2], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+1], NULL );
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+2], NULL );

    [vTaskStartScheduler()];
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for ( ;; ) {
        vPrintString( "Ejecución única\r\n" );
        [vTaskDelete( NULL )];
    }
}
```

```
void vTask2 ( void *pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = [pdMS_TO_TICKS]( 300 );

    xLastWakeTime = [xTaskGetTickCount()];
    for ( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [vTaskDelayUntil]( [&xLastWakeTime, xPeriod] );
    }
}
```

Pregunta **11**Parcialmente
correctaPuntúa 0,43
sobre 1,00

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Vincular vTask1 y vTask2 a través de un mecanismo de sincronización.
- 2- vTask2 debe cambiar el estado del LED verde cuando vTask1 finaliza un ciclo de espera activa.
- 3- vTask2 espera indefinidamente la señal de sincronización.

```
SemaphoreHandle_t xMutex;
SemaphoreHandle_t xBinarySemaphore;

int main( void )
{
    xMutex  = xSemaphoreCreateMutex()  ;

    /*Creación de tareas, inicio del planificador*/

    for( ;; );

    return 0;
}

void vTask1 ( void *pvParameters )
{
    for ( ;; ) {
        for ( i=0; i<DELAY_LOOP_COUNT; i++) { vPrintString( "Espera activa\r\n" ); }
        xSemaphoreGive  ( xMutex, portMAX_DELAY  );
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = pdMS_TO_TICKS  ( 1000 );

    {
        for ( ;; ) {
            if( xSemaphoreTake  ( xMutex, xDelay  ) == pdTRUE) {
                Board_LED_Toggle(5); //LED verde;
            }else {
                Board_LED_Toggle(4); //LED rojo;
```

```
}  
}  
}
```

xMutex, xDelay	xSemaphoreCreateBinary()	xDelay
pdMS_TO_TICKS	xBinarySemaphore	xMutex
xSemaphoreTakeFromISR	xSemaphoreGiveFromISR	xSemaphoreGive
xSemaphoreTake	xHigherPriorityTaskWoken	xSemaphoreCreateMutex()
xMutex, portMAX_DELAY	portYIELD_FROM_ISR	xBinarySemaphore, xDelay
xBinarySemaphore, portMAX_DELAY		

Respuesta parcialmente correcta.

Ha seleccionado correctamente 3.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- Vincular vTask1 y vTask2 a través de un mecanismo de sincronización.
- 2- vTask2 debe cambiar el estado del LED verde cuando vTask1 finaliza un ciclo de espera activa.
- 3- vTask2 espera indefinidamente la señal de sincronización.

```
SemaphoreHandle_t xMutex;  
SemaphoreHandle_t xBinarySemaphore;  
int main( void )  
{  
    [xBinarySemaphore] = [xSemaphoreCreateBinary()];  
  
    /*Creación de tareas, inicio del planificador*/  
  
    for( ;; );  
  
    return 0;  
}
```

```
    }

void vTask1 ( void *pvParameters )
{
    for ( ;; ) {
        for ( i=0; i<DELAY_LOOP_COUNT; i++) { vPrintString( "Espera activa\r\n" ); }
        [xSemaphoreGive] ([xBinarySemaphore]);
    }
}

void vTask2 ( void *pvParameters )
{
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );
    {
        for ( ;; ) {
            if( [xSemaphoreTake]( [xBinarySemaphore, portMAX_DELAY] ) == pdTRUE) {
                Board_LED_Toggle(5); //LED verde;
            }else {
                Board_LED_Toggle(4); //LED rojo;
            }
        }
    }
}
```


Pregunta **12**Parcialmente
correctaPuntúa 0,57
sobre 0,71

Seleccione las afirmaciones correctas respecto a serialización del evento A respecto del evento B.

Seleccione una o más de una:

- ☐ a. El evento A debe esperar la ocurrencia del evento B, e inversamente.
- ☐ b. Para forzar la situación de serialización se puede usar un barrier, inicialmente en 0.
- ☐ c. Para forzar la situación de serialización se puede usar un barrier, inicializado en 1.
- ☒ d. El evento A debe esperar la ocurrencia del evento B.
- ☐ e. Para forzar la situación de serialización se puede usar un mutex, inicialmente en 1.
- ☒ f. Para forzar la situación de serialización se puede usar un mutex, inicialmente en 0.
- ☒ g. El evento A y el evento B no deben ser concurrentes.



Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

Las respuestas correctas son: El evento A debe esperar la ocurrencia del evento B., Para forzar la situación de serialización se puede usar un mutex, inicialmente en 0.

[◀ Parcial 2](#)

[Área personal](#) / [Mis cursos](#) / [Técnicas Digitales III \(Práctica y Exámenes\) 2021](#) / [Parcial 2](#) / [Recuperatorio 2](#)

Comenzado el	Thursday, 2 de September de 2021, 19:00
Estado	Finalizado
Finalizado en	Thursday, 2 de September de 2021, 19:52
Tiempo empleado	52 minutos 24 segundos
Puntos	7,66/10,00
Calificación	7,66 de 10,00 (77%)



Pregunta **1**

Correcta

Puntúa 0,71
sobre 0,71

Seleccione las afirmaciones correctas respecto a serialización del evento A respecto del evento B.

Seleccione una o más de una:

- ☐ a. El evento A y el evento B no deben ser concurrentes.
- ☐ b. Para forzar la situación de serialización se puede usar un barrier, inicializado en 1.
- ☐ c. Para forzar la situación de serialización se puede usar un barrier, inicialmente en 0.
- ☒ d. Para forzar la situación de serialización se puede usar un mutex, inicialmente en 0. 
- ☐ e. El evento A debe esperar la ocurrencia del evento B, e inversamente.
- ☐ f. Para forzar la situación de serialización se puede usar un mutex, inicialmente en 1.
- ☒ g. El evento A debe esperar la ocurrencia del evento B. 

Respuesta correcta

Las respuestas correctas son: El evento A debe esperar la ocurrencia del evento B., Para forzar la situación de serialización se puede usar un mutex, inicialmente en 0.

Pregunta **2**Parcialmente
correctaPuntúa 0,36
sobre 0,72

Seleccione las afirmaciones correctas respecto a la implementación foreground / background (primer plano / segundo plano) para RTS.

Seleccione una o más de una:

- ☐ a. Para solucionar posibles problemas de incoherencia de datos se puede hacer uso de semáforos o mutex.
- ☒ b. La ventaja de estos sistemas respecto de bucle scan es que son más sencillos de programar. ✗
- ☒ c. La latencia en estos sistemas está definida por las rutinas de atención de interrupción. ✓
- ☒ d. Los sistemas foreground/background tienen mayor eficiencia que los sistemas scan loop. ✗
- ☐ e. La latencia en estos sistemas está definida por el tiempo que demora en ejecutar el bucle de foreground.

Respuesta parcialmente correcta.

Ha seleccionado demasiadas opciones.

La respuesta correcta es: La latencia en estos sistemas está definida por las rutinas de atención de interrupción.


Pregunta **3**

Correcta

Puntúa 0,71
sobre 0,71

Seleccione qué afirmación es correcta respecto a administración de memoria con mapas de bits, para particiones variables.

Seleccione una o más de una:

- ☒ a. La fragmentación externa no depende del tamaño de la unidad de asignación de memoria. 
- ☐ b. La asignación y liberación de memoria demoran el mismo tiempo.
- ☐ c. Cuanto menor sea la unidad de asignación, mayor será la fragmentación interna.
- ☐ d. Es rápida la asignación de memoria y lenta la liberación de memoria.
- ☐ e. Cuanto menor sea la unidad de asignación de memoria, menor será el mapa de bits.

Respuesta correcta

La respuesta correcta es: La fragmentación externa no depende del tamaño de la unidad de asignación de memoria.

Pregunta 4

Parcialmente
correctaPuntúa 0,58
sobre 1,00

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

1- La tarea vTask1 debe ejecutarse una única vez y debe ser la primera en ejecutarse.

2- vTask2 debe ser la primer tarea en crearse.

3- vTask2 debe pasar a estado "Lista" exactamente cada 300ms y cambiar el estado del LED verde.

```
int main( void )
{
    xTaskCreate ( vTask2 , NULL, configMINIMAL_STACK_SIZE, NULL,
    tskIDLE_PRIORITY+2 , NULL );
    xTaskCreate ( vTask1 , NULL, configMINIMAL_STACK_SIZE, NULL,
    tskIDLE_PRIORITY+1 , NULL );
    vTaskStartScheduler();
    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    for ( ;; ) {
        vPrintString( "Ejecución única\r\n" );
        vTaskDelete( NULL );
    }
}

void vTask2 ( void *pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = pdMS_TO_TICKS ( 300 );
    xLastWakeTime = ;
    for ( ;; ) {
```

```

Board_LED_Toggle(5); //LED verde;
    vTaskDelay ( xPeriod );
}
}

```

xTaskHandle	tskIDLE_PRIORITY+1	void vTask2	vTaskPrioritySet
void vTask1	pdMS_TO_TICKS	xPeriod	xTaskCreate
vTaskDelayUntil	tskIDLE_PRIORITY+2	vTaskDelay	vTask2
vTaskStartScheduler()	&xLastWakeTime, xPeriod	xTaskGetTickCount()	vPrintString
300	for (;;)	vTaskDelete(NULL)	pvParameters
xLastWakeTime	xDelay	vTask1	

Respuesta parcialmente correcta.

Ha seleccionado correctamente 7.

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe ejecutarse una única vez y debe ser la primera en ejecutarse.
- 2- vTask2 debe ser la primer tarea en crearse.
- 3- vTask2 debe pasar a estado "Lista" exactamente cada 300ms y cambiar el estado del LED verde.

```

int main( void )
{
    [xTaskCreate]( [vTask2], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+1], NULL );
    [xTaskCreate]( [vTask1], NULL, configMINIMAL_STACK_SIZE, NULL, [tskIDLE_PRIORITY+2], NULL );

    [vTaskStartScheduler()];
    for( ;; );
}

```

```
        return 0;
    }

void vTask1 ( void *pvParameters )
{
    for ( ;; ) {
        vPrintString( "Ejecución única\r\n" );
        [vTaskDelete( NULL )];
    }
}

void vTask2 ( void *pvParameters )
{
    TickType_t xLastWakeTime;
    const TickType_t xPeriod = [pdMS_TO_TICKS]( 300 );





    xLastWakeTime = [xTaskGetTickCount()];
    for ( ;; ) {
        Board_LED_Toggle(5); //LED verde;
        [vTaskDelayUntil]( [&xLastWakeTime, xPeriod] );
    }
}
```


Pregunta **5**

Incorrecta

Puntúa 0,00
sobre 0,73

La inversión de prioridad:

- ☒ a. Se puede dar entre dos tareas de distinta prioridad que utilicen un semáforo binario compartido. 
- ☒ b. Puede ser evitada utilizando un mutex en lugar de un semáforo binario en FreeRTOS. 
- ☐ c. Se puede dar entre dos tareas de distinta prioridad que utilicen una cola de mensajes compartida.
- ☒ d. Siempre aumenta la latencia de la tarea más prioritaria. 
- ☒ e. Intercambia las prioridades de las tareas involucradas, quedando con menos prioridad la tarea que era más prioritaria. 

Respuesta incorrecta.

Las respuestas correctas son: Se puede dar entre dos tareas de distinta prioridad que utilicen un semáforo binario compartido., Se puede dar entre dos tareas de distinta prioridad que utilicen una cola de mensajes compartida., Siempre aumenta la latencia de la tarea más prioritaria.

Pregunta **6**Parcialmente
correctaPuntúa 0,75
sobre 1,00

Un proceso posee 3 hilos: main, hilo0, hilo1. Los 3 hilos incrementan las variables total y total1.

- 1- El hilo main crea el hilo1.
- 2- El hilo main debe esperar al hilo1 antes de crear el hilo0.
- 3- El hilo main crea al hilo 0.
- 4- El hilo main debe esperar al hilo0 antes de acceder a las variables total y total1.
- 5- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
```

```
pthread_t h[2];
int a[5]={0}, total, total1;
```

```
void *hilo0 (void * nro) {
int d, t;
    d = *(int*)nro;
    //línea en blanco ✓
    for(t=0; t < 100000 ; t++){
        total=total+d;
        total1=total1+a[2];
    }
    pthread_exit(NULL);
}
```

```
void *hilo1 (void * nro) {
int d, t;
    d = *(int*)nro;
```

```
    //línea en blanco ✓
```

```
        for(t=0; t < 10000 ; t++){
            total=total+a[3];
            total1=total1+d;
        }
        pthread_exit(NULL);
    }
int main() {
    int j;
    total=10000;
    total1=100000;
    a[0]=10;
    a[1]=1;
    a[3]=2;
    //linea en blanco ✓
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
    pthread_join(h[1],NULL); ✓
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    pthread_join(h[0],NULL); ✓
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+a[1];
    }
    //linea en blanco ✓
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}
```

/*

En consola vemos

Total=

210000

✗

,Total1=

3200000

✗

*/

//linea en blanco

pthread_mutexattr_init(&mtxattr);

pthread_join(h[1],NULL);

11200000

3200000

pthread_detach(pthread_self());

210000

1210000

1230000

pthread_join(h[2],NULL);

pthread_yield();

pthread_exit(NULL);

pthread_join(h[0],NULL);

Respuesta parcialmente correcta.

Ha seleccionado correctamente 6.

La respuesta correcta es:

Un proceso posee 3 hilos: main, hilo0, hilo1. Los 3 hilos incrementan las variables total y total1.

1- El hilo main crea el hilo1.

2- El hilo main debe esperar al hilo1 antes de crear el hilo0.

3- El hilo main crea al hilo 0.

4- El hilo main debe esperar al hilo0 antes de acceder a las variables total y total1.

5- El hilo main antes de terminar muestra el valor de las variables total y total1.

Completar:

#include <pthread.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

```
pthread_t h[2];
int a[5]={0}, total, total1;

void *hilo0 (void * nro) {
int d, t;
    d = *(int*)nro;
    [//linea en blanco]
    for(t=0; t < 100000 ; t++){
        total=total+d;
        total1=total1+a[2];
    }
    pthread_exit(NULL);
}

void *hilo1 (void * nro) {
int d, t;
    d = *(int*)nro;
    [//linea en blanco]
    for(t=0; t < 10000 ; t++){
        total=total+a[3];
        total1=total1+d;
    }
    pthread_exit(NULL);
}

int main() {
int j;
    total=10000;
    total1=100000;
    a[0]=10;
    a[1]=1;
    a[3]=2;
```

```
    [//linea en blanco]
    pthread_create(&h[1], NULL, hilo1, (void *) (&a[1]) );
    [pthread_join(h[1],NULL);]
    pthread_create(&h[0], NULL, hilo0, (void *) (&a[0]) );
    [pthread_join(h[0],NULL);]
    for(j=0; j < 100000 ; j++){
        total=total+a[3];
        total1=total1+a[1];
    }
    [//linea en blanco]
    printf("Total= %d, Total1= %d\n",total,total1);
    pthread_exit(NULL);
}
```

```
/*
En consola vemos
Total= [1230000] ,Total1= [210000]
*/
```

Pregunta **7**

Correcta

Puntúa 1,00
sobre 1,00

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe enviar a través de una cola de mensajes FIFO el valor entero 123 de manera continua.
- 2- En el caso de que la cola esté llena, vTask1 se debe bloquear durante 1 segundo.
- 3- La tarea vTask2 debe recibir los datos enviados por vTask1.
- 4- En el caso de que la cola esté vacía, vTask2 se debe bloquear durante 500ms.

```
QueueHandle_t xQueue;
int main( void )
{
    UBaseType_t uxQueueLength = 10;
    UBaseType_t uxItemSize = sizeof( int32_t );

    xQueue = xQueueCreate ( uxQueueLength, uxItemSize );

    /*Creación de tareas, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    int32_t lValueToSend = 123;
    const TickType_t xDelay = pdMS_TO_TICKS ( 1000 );

    for ( ;; ) {
        xQueueSendToBack ( xQueue, &lValueToSend, xDelay );
    }
}

void vTask2 ( void *pvParameters )
{
    int32_t lReceivedValue;
```

```

const TickType_t xDelay =  ✓ ( 500 );

for ( ;; ) {
     ✓ ( xQueue,  ✓ );
    vPrintString( "Dato recibido\r\n" );
}

```

<input type="text" value="xQueueReceiveFromISR"/>	<input type="text" value=" &lValueToSend, xDelay"/>	<input type="text" value=" xDelay"/>
<input type="text" value="uxItemSize"/>	<input type="text" value=" &lValueToSend, portMAX_DELAY"/>	<input type="text" value=" lReceivedValue"/>
<input type="text" value="uxQueueLength, uxItemSize"/>	<input type="text" value=" xHigherPriorityTaskWoken"/>	<input type="text" value=" xQueueReceive"/>
<input type="text" value="portMAX_DELAY"/>	<input type="text" value=" xQueueSendToBackFromISR"/>	<input type="text" value=" xQueueCreate"/>
<input type="text" value="portYIELD_FROM_ISR"/>	<input type="text" value=" &lReceivedValue, portMAX_DELAY"/>	<input type="text" value=" uxQueueLength"/>
<input type="text" value=" &lReceivedValue, xDelay"/>	<input type="text" value=" pdMS_TO_TICKS"/>	<input type="text" value=" xQueueSendToBack"/>
<input type="text" value=" xQueueSendToFront"/>	<input type="text" value=" lValueToSend"/>	<input type="text" value=" xQueue"/>
<input type="text" value=" xQueueSendToFrontFromISR"/>	<input type="text" value=" &lValueToSend"/>	

Respuesta correcta

La respuesta correcta es:

Complete la aplicación FreeRTOS para que cumpla las siguientes consignas:

- 1- La tarea vTask1 debe enviar a través de una cola de mensajes FIFO el valor entero 123 de manera continua.
- 2- En el caso de que la cola esté llena, vTask1 se debe bloquear durante 1 segundo.
- 3- La tarea vTask2 debe recibir los datos enviados por vTask1.

4- En el caso de que la cola esté vacía, vTask2 se debe bloquear durante 500ms.

```
QueueHandle_t xQueue;

int main( void )
{
    UBaseType_t uxQueueLength = 10;
    UBaseType_t uxItemSize = sizeof( int32_t );

    xQueue = [xQueueCreate]( [uxQueueLength, uxItemSize] );

    /*Creación de tareas, inicio planificador*/

    for( ;; );
    return 0;
}

void vTask1 ( void *pvParameters )
{
    int32_t lValueToSend = 123;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 1000 );

    for ( ;; ) {
        [xQueueSendToBack]( xQueue, [&lValueToSend, xDelay] );
    }
}

void vTask2 ( void *pvParameters )
{
    int32_t lReceivedValue;
    const TickType_t xDelay = [pdMS_TO_TICKS]( 500 );

    for ( ;; ) {
        [xQueueReceive]( xQueue, [&lReceivedValue, xDelay] );
        vPrintString( "Dato recibido\r\n" );
    }
}
```

Pregunta **8**Parcialmente
correctaPuntúa 0,24
sobre 0,71

Indique características del algoritmo shortest process next.

Seleccione una o más de una:

- ☐ a. No es necesario tener todos los procesos en tiempo inicial T0.
- ☐ b. Es una mejora al algoritmo de Round-Robin.
- ☐ c. El cálculo de envejecimiento tiene alto costo computacional.
- ☐ d. Es facil de implementar, sumando el nuevo valor a la estimación actual y desplazando un bit a la derecha.
- ☒ e. Usa una aproximación basada en registración del comportamiento anterior.



Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.

Las respuestas correctas son: Usa una aproximación basada en registración del comportamiento anterior., Es facil de implementar, sumando el nuevo valor a la estimación actual y desplazando un bit a la derecha., No es necesario tener todos los procesos en tiempo inicial T0.

Pregunta **9**

Correcta

Puntúa 1,00
sobre 1,00**ENUNCIADO**

Modifique el archivo padre.c para que envíe al proceso hijo la cadena de caracteres "HOLA_HIJO" mediante la cola de mensajes declarada como MQ_PATH.

Por el lado del padre:

1. El padre debe abrir la cola de mensajes MQ_PATH en modo de solo escritura/lectura.
2. El padre debe enviar al proceso hijo la cadena de caracteres "HOLA_HIJO" mediante la cola de mensajes declarada como MQ_PATH.
3. Se recomienda que el padre elimine la cola de mensajes luego de escribir el mensaje en la cola.

Por el lado del hijo:

1. El proceso hijo recibe como parámetro la variable mqd.
2. Lee de la cola de mensajes lo enviado por el proceso padre y cierra la cola de mensaje.
3. Solo si recibe "HOLA_HIJO" escribe por consola el resultado del ejercicio.

PASOS A SEGUIR

1. Descargue los siguientes 2 archivos en un mismo directorio:

- 1.a. Plantilla del código fuente del proceso padre:

padre.c : <https://nube.ingenieria.uncuyo.edu.ar/s/A2YXLEn4krGdTwB>

- 1.b. Archivo objeto del proceso hijo para arquitectura de 64 bits:

hijo.o : <https://nube.ingenieria.uncuyo.edu.ar/s/9g76HDmcdig8cY4>

2. Complete el archivo padre.c según el enunciado del ejercicio.

3. Copie y pegue en consola el siguiente comando para compilar ambos archivos:

```
gcc -c padre.c -lrt && gcc hijo.o padre.o -o padre -lrt
```

4. El binario generado por el comando anterior se ejecuta en consola con:

```
./padre
```

5. El proceso hijo imprime por consola:

```
Hijo en ejecucion...
```

```
Hijo: mensaje leído es HOLA_HIJO
```

```
Hijo: el resultado del ejercicio es *** XXX ***
```

```
Hijo: cola de mensajes cerrada.
```

6. Si el proceso hijo no recibe "HOLA_HIJO", imprime por consola:

```
Hijo: mensaje leído no coincide con HOLA_HIJO
```

RESPUESTA DEL EJERCICIO

La respuesta de este ejercicio es el número de 3 cifras XXX que imprime el proceso hijo por consola. Copie este número de 3 cifras XXX en la casilla de abajo.

Respuesta:



La respuesta correcta es: 347

Pregunta **10**

Correcta

Puntúa 0,71
sobre 0,71

Seleccione qué afirmación es correcta respecto a la técnica de paginación para memoria virtual.

Seleccione una o más de una:

- ☐ a. Al producirse un fallo de página siempre se debe ejecutar un algoritmo de reemplazo de páginas.
- ☒ b. El bus de direcciones del procesador no está directamente conectado a la memoria física. ✓
- ☐ c. Se crea un segmento de código, datos y pila para cada programa en ejecución.
- ☒ d. Permite ejecutar programas más grandes que la memoria física. ✓
- ☐ e. Se divide la memoria física en páginas de igual tamaño, que es potencia de 2.

Respuesta correcta

Las respuestas correctas son: Permite ejecutar programas más grandes que la memoria física., El bus de direcciones del procesador no está directamente conectado a la memoria física.

Pregunta **11**Parcialmente
correctaPuntúa 0,89
sobre 1,00

En el programa el hilo main crea 150 hilos.

- 1- El hilo main y el resto de los hilos realizan una suma en la variable total y se sincronizan con un mutex.
- 2- El hilo main espera a que los hilos terminen, muestra el valor de la variable total y luego termina él.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
```

```
int total, a[5]={0};
pthread_t h[150];
```

```
pthread_mutex_t b=PTHREAD_MUTEX_INITIALIZER;
```



```
void *hilo1 () {
int t;
```

```
//linea en blanco
```



```
for(t=0; t < 10000 ; t++){
```

```
pthread_mutex_lock(&b);
```



```
total=total+a[0];
```

```
pthread_mutex_unlock(&b);
```

```
}
```

```
//linea en blanco
```



```
pthread_exit(NULL);
```

```
}
```

```
int main() {
```

```
int t;
```

```
total=10000000;
```

```
a[0]=2;
```

```
a[1]=1;
```

```
for(t=0; t< 150 ; t++){
```

```
pthread_create(&h[t], NULL, hilo1, NULL);
```



```
}
```

```
//linea en blanco
```



```
for(t=0; t < 10000 ; t++){
```

```
pthread_mutex_lock(&b);
```



```
total=total+a[1];
```

```
pthread_mutex_unlock(&b);
```

```
}
```

```
for(t=0; t< 150 ; t++){
    pthread_create(&h[t], NULL, hilo1, NULL);
}
printf("total= %d\n",total);
pthread_exit(NULL);
}
*/
/*
En consola vemos
Total= 40100000
*/
```

pthread_join(h[t],NULL);

pthread_mutex_lock(&b);

pthread_mutex_t b;

pthread_create(&h[t], NULL, hilo1, NULL);

pthread_create(&a[t], NULL, hilo0, (void *) (&v[t]))

pthread_mutex_unlock(&b);

pthread_join(h[1],NULL);

pthread_mutex_t b=PTHREAD_MUTEX_INITIALIZER;

pthread_detach(h[t],NULL)

//linea en blanco

40100000

4020000

Respuesta parcialmente correcta.

Ha seleccionado correctamente 8.

La respuesta correcta es:

En el programa el hilo main crea 150 hilos.

1- El hilo main y el resto de los hilos realizan una suma en la variable total y se sincronizan con un mutex.

2- El hilo main espera a que los hilos terminen, muestra el valor de la variable total y luego termina él.

Completar

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int total, a[5]={0};
pthread_t h[150];
[pthread_mutex_t b=PTHREAD_MUTEX_INITIALIZER;]

void *hilo1 () {
int t;
    [//linea en blanco]
    for(t=0; t < 10000 ; t++){
        [pthread_mutex_lock(&b);]
        total=total+a[0];
        pthread_mutex_unlock(&b);
    }
    [//linea en blanco]
    pthread_exit(NULL);
}

int main() {
int t;
    total=1000000;
    a[0]=2;
    a[1]=1;
    for(t=0; t< 150 ; t++){
        [pthread_create(&h[t], NULL, hilo1, NULL);]
    }
    [//linea en blanco]
    for(t=0; t < 10000 ; t++){
        [pthread_mutex_lock(&b);]
        total=total+a[1];
        pthread_mutex_unlock(&b);
    }
    for(t=0; t< 150 ; t++){
        [pthread_join(h[t],NULL);]
    }
    printf("total= %d\n",total);
}
```

```
pthread_exit(NULL);  
}  
*/  
/*  
En consola vemos  
Total=[4010000]  
*/
```

Pregunta **12**

Correcta

Puntúa 0,71
sobre 0,71

De las siguientes afirmaciones, seleccione la o las que crea que son verdaderas respecto al uso de hilos.

Seleccione una o más de una:

- ☐ a. Ante un error en tiempo de ejecución en el código del hilo, solo éste termina y su proceso asociado se sigue ejecutando.
- ☒ b. Permiten el paralelismo real en computadoras de mas de un núcleo. ✓
- ☒ c. Ante un error en tiempo de ejecución en el código del hilo, se termina el proceso al que perternece el hilo. ✓
- ☐ d. La creación y destrucción de hilos es más rápida que la de subrutinas.
- ☐ e. Todas las afirmaciones solo son válidas para sistemas multiprocesador o multinúcleo.

Respuesta correcta

Las respuestas correctas son: Ante un error en tiempo de ejecución en el código del hilo, se termina el proceso al que perternece el hilo., Permiten el paralelismo real en computadoras de mas de un núcleo.

[◀ Parcial 2](#)[Ir a...](#)