



Examen Final de Sistemas Operativos
16 de septiembre de 2008 a las 16:00 horas
Aulas 2.3.B03, 2.3.B04, 2.3.B05

NOTAS:

- * La fecha de publicación de las notas, así como de revisión se notificarán por Aula Global
 - * Para la realización del presente examen se dispondrá de **1 hora y 45 minutos**.
 - * **El examen se contesta en las hojas dadas con el enunciado.**
 - * **No** se pueden utilizar libros **ni** apuntes, ni usar móvil (o similar)
 - * Será necesario presentar el DNI o carnet universitario para realizar la entrega del examen
-

Ejercicio 1 (2 puntos). Conteste a las siguientes preguntas:

- a. ¿Qué es el estado del procesador?
- b. ¿Cuáles son los segmentos (regiones) básicos del modelo de memoria de un proceso?
Indica además sus características.
- c. ¿De qué se compone la estructura de un sistema de archivos en UNIX? Indica que contiene cada una de ellas.
- d. ¿Cuál es la diferencia entre enlace físico y enlace simbólico?

Ejercicio 2 (3 puntos). Realizar una aplicación que esté compuesta por un programa que debe crear 2 hijos.

Tanto el padre como los 2 hijos deben llamar a la función

int calcularTemperatura (int t)

La función *calcularTemperatura* no debe ser realizada por el alumno. Esta función devuelve la temperatura calculada y debe ser invocada, la primera vez pasándole el parámetro 0 y las siguientes pasándole el dato que ella misma devolvió en la invocación anterior.

La función *calcularTemperatura* devuelve un valor entre 0 y 99 con la temperatura calculada (cuantas más llamadas se realicen a la función mas precisa será ésta).

El proceso padre debe crear los 2 hijos y esperar 3 segundos. Durante la espera el padre también procederá a invocar repetidamente a la función *calcularTemperatura*. Cuando hayan pasado los 3 segundos el padre finalizará la realización de su cálculo de Temperatura y mandará la señal *SIGUSR1* a los hijos para indicarles que deben finalizar.

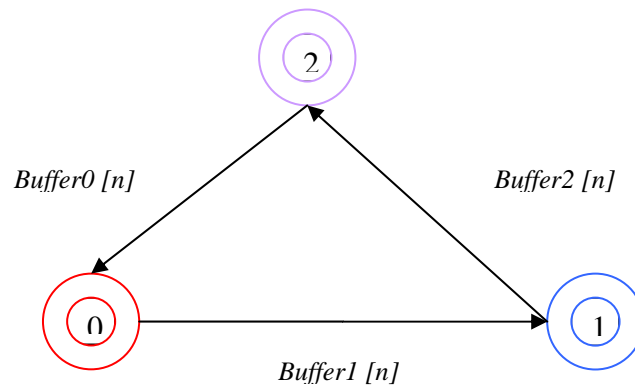
Después esperará su finalización y recogerá el dato de temperatura que los hijos le pasarán utilizando la orden *exit*.

Los datos recogidos serán utilizados para calcular la temperatura final realizando la media entre las 3 obtenidas (la del padre y la de los 2 hijos). Esta media que será mostrada en pantalla.

Los hijos, que estarán invocando repetidamente a la función *calcularTemperatura*, cuando llegue la señal *SIGUSR1*, deberán devolver el último valor retornado por la función *calcularTemperatura*, utilizando la función *exit()*



Ejercicio 3 (3 puntos). Dado el siguiente esquema de productor consumidor enlazado entre 3 procesos:



Se describen las siguientes características:

- Todos los procesos son productores y a la vez consumidores. De esta forma, un productor no puede escribir en un buffer lleno y un consumidor no puede leer de un buffer vacío.
- Un proceso primero Produce y después consume. Si no produce se bloquea hasta que pueda producir; después consumirá y se bloqueará hasta que consuma. Después intentará de nuevo producir y así de forma indefinida (mediante un bucle infinito)
- Los 3 procesos comparten tres buffers (dos a dos) todos ellos del mismo tamaño. Cada uno de los procesos usan el buffer siguiente ($i+1$) y el anterior (i). Cada proceso(i) lee del **buffer i** un elemento y lo inserta en el **buffer $i+1$** .
- Los buffers se encuentran aleatoriamente llenos (con valores entre 0 y n) siendo siempre la suma menor que 3 veces el tamaño de un buffer (siempre habrá algún buffer que no esté del todo lleno).

A continuación se muestra el programa principal. Se deberá escribir el código asociado a la función `ProductorConsumidor` y a la definición de los mutex, variables condicionales y variables compartidas:

/ Mutex, variables condicionales y variables compartidas */*

.....

/ Código Productor Consumidor */*

.....

/ Programa Principal */*

```

main(int argc, char *argv[]){
    pthread_t *arrayThread;
    int i;

    for (i=0; i<NTHREADS;i++){
        pthread_mutex_init(&mutex[i], NULL);
        pthread_cond_init(&no_lleno[i], NULL);
        pthread_cond_init(&no_vacio, NULL);
    }
  
```



*/*inicializa buffers. Asigna llena de forma aleatoria los 3 buffers. Se garantiza que al menos uno de ellos no está lleno*/*

inicializa (buffer); //No es necesario definir esta funcion

/ Array de ProductoresConsumidores */*

arrayThread = (pthread_t *)malloc(sizeof(pthread_t) * NTHREADS);

for (i=0; i<NTHREADS;i++){

pthread_create(&arrayThread[i], NULL, ProductorConsumidor, &i);
}

for (i=0; i<NTHREADS;i++){

pthread_join(arrayThread[i], NULL);
}

for (i=0; i<NTHREADS;i++){

pthread_mutex_destroy(&mutex[i]);
pthread_cond_destroy(&no_lleno[i]);
pthread_cond_destroy(&no_vacio[i]);
}

return 0;

}

Solucion:

#define MAX_BUFFER 1024 /* tamaño del buffer */

#define NTHREADS 3

pthread_t arrayThreads [NTHREADS];

pthread_mutex_t mutex[NTHREADS]; /* mutex control acceso buffer compartido */

pthread_cond_t buffer_no_lleno[NTHREADS]; /* controla el llenado del buffer */

pthread_cond_t buffer_no_vacio[NTHREADS]; /* controla el vaciado del buffer */

int n_elementos[NTHREADS]; /* número de elementos en el buffer */

int posicion [NTHREADS]; /* puntero de posición cada buffer */

int buffer[NTHREADS][MAX_BUFFER]; /* buffer común */

/ código del ProductorConsumidor */*

void ProductorConsumidor (int * i) {

int dato;

for(;;){

/ producir dato del buffer [(i+1)%NTHREADS] */*

dato = random(1000);

pthread_mutex_lock(&mutex[(i+1)%NTHREADS]); */* acceder al buffer */*

while (n_elementos[(i+1)%NTHREADS] == MAX_BUFFER) */* si buffer lleno */*

pthread_cond_wait(&buffer_no_lleno[(i+1)%NTHREADS], &mutex[(i+1)%NTHREADS]);

/ se bloquea */*

buffer[(i+1)%NTHREADS][posicion[(i+1)%NTHREADS]] = dato;



```

posicion[(i+1)%NTHREADS] = (posicion[i+1)%NTHREADS] + 1) % MAX_BUFFER;
n_elementos[(i+1)%NTHREADS] ++;
pthread_cond_signal(&buffer_no_vacio[(i+1)%NTHREADS]); /* buffer no vacio */
pthread_mutex_unlock(&buffer_mutex[(i+1)%NTHREADS]);

/* producir dato del buffer [i] */

pthread_mutex_lock(&mutex[i]); /* acceder al buffer */
while (n_elementos[i] == 0) /* si buffer vacio */
    pthread_cond_wait(&buffer_no_vacio[i], &mutex[i]); /* se bloquea */
dato = buffer[i][posicion[i]];
posicion[i] = (posicion[i] + 1) % MAX_BUFFER;
n_elementos --;
pthread_cond_signal(&buffer_no_lleno[i]); /* buffer no lleno */
pthread_mutex_unlock(&mutex[i]);
printf("Consume %d \n", dato); /* consume dato */
}
pthread_exit(0);

}
pthread_exit(0);
}

/* Programa Principal */

main(int argc, char *argv[]){
    pthread_t * arrayThread;
    int i;

    for (i=0; i<NTHREADS;i++){
        pthread_mutex_init(&mutex[i], NULL);
        pthread_cond_init(&no_lleno[i], NULL);
        pthread_cond_init(&no_vacio, NULL);
    }

    /*inicializa buffers. Asigna llena de forma aleatoria los 3 buffers. Se garantiza que al menos
    uno de ellos no está lleno.*/

    inicializa (buffer); //No es necesario definir esta funcion

    /* Array de ProductoresConsumidores */
    arrayThreads = (pthread_t *)malloc( sizeof(pthread_t) * NTHREADS);

    for (i=0; i<NTHREADS;i++){
        pthread_create(&arrayThread[i], NULL, ProductorConsumidor, &i);
    }

    for (i=0; i<NTHREADS;i++){
        pthread_join(arrayThread[i], NULL);
    }

    for (i=0; i<NTHREADS;i++){
        pthread_mutex_destroy(&mutex[i]);
        pthread_cond_destroy(&no_lleno[i]);
        pthread_cond_destroy(&no_vacio[i]);
    }

    return 0;
}

```

**Ejercicio 4 (2 puntos).**

a.- (1 punto) Determine y especifique detalladamente el número de accesos físicos a disco necesarios, como mínimo, en un sistemas UNIX, para ejecutar la siguiente operación:

```
fd = open ("lib/agenda/direcciones", RD_ONLY);
```

Suponga que la cache del sistema de archivos está inicialmente vacía. Y que el bloque del directorio de trabajo está cargado en memoria.

b.- (1 punto) ¿Cuál de las siguientes afirmaciones es falsa? ¿Por qué? a)

- a) El número de enlaces de un archivo en UNIX se almacena en la entrada de directorios correspondiente
- b) Crear un enlace simbólico a un archivo incrementa el número de i-nodos ocupados en el sistema
- c) Crear un enlace físico a un archivo incrementa el número de enlaces del archivo.
- d) Todos los archivos de un sistema de archivos determinados utilizan el mismo tamaño de bloque

OBSERVACION: Si la respuesta no se razona, no se puntuará aunque la opción señalada sea la correcta.

Solución:

Parte a.- Suponiendo que los directorios son pequeños, como mínimo habrá que hacer 6 accesos. Éstos son:

- Traer el primer bloque del directorio de trabajo para conocer el i-nodo ./lib
- Leer el i-nodo de ./lib
- Traer el primer bloque del directorio ./lib para conocer el i-nodo de agenda
- Leer el i-nodo agenda
- Traer el primer bloque del fichero ./lib/agenda para conocer el i-nodo de direcciones
- Leer el i-nodo de ./lib/agenda/direcciones

Parte b. La respuesta falsa es la A. En efecto, el número de enlaces de un archivo es un valor que se almacena en el i-nodo del archivo y no en la entrada de directorio. La entrada de directorio correspondiente solo almacena el nombre del archivo y su número de i-nodo. A partir de ese número de i-nodo, el sistema de archivos es capaz de localizar el i-nodo correspondiente.