SISTEMAS OPERATIVOS - TEORÍA
16 de Enero de 2024

Nombre _____ DNI _____

Apellidos _____ Grupo _____

Cuestión 1 (2 puntos). Considere un sistema de ficheros UNIX que utiliza bloques de 4096 bytes y punteros y direcciones de disco de 32 bits. Los nodos-i contienen una entrada directa del índice, una indirecta simple y una indirecta doble.

a) (**0.5 puntos**) Calcule el tamaño máximo de un fichero en dicho sistema.

b) (**0.75 puntos**) Dado el siguiente grafo de un árbol de directorios, en el que los óvalos representan directorios y los rectángulos ficheros regulares, complete la información que falta en el grafo, la tabla de nodos-i, el mapa de bits y el contenido de los bloques utilizados.

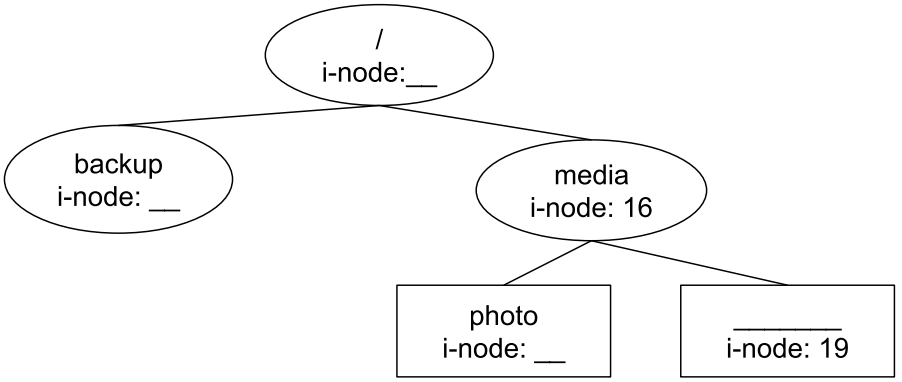

Tabla de nodos-i

nodo-i	2	3		18	19
Tamaño (en bloques)	1		1	1	3
Tipo		D	D		F
Enlaces	4		2		1
Directo	1	2	3	6	8
Ind. Simple	-	-	-	-	4
Ind. Doble	-	-	-	-	-

Mapa de bits:

Bloques utilizados:

1		2		3		4	6	8	9	10
. .. media backup	2 2 16 3	. ..	3 2	. .. photo video	16 18 19		Datos de usuario		Datos de usuario	Datos de usuario

c) (**0.75 puntos**) Describa los cambios que se producirán en las estructuras del apartado anterior cuando se ejecuten los siguientes comandos.

$ ln /media/video /backup/video_backup

$ ln -s /media/photo /backup/photo_backup

$ mv /backup/photo_backup /media/video2

Cuestión 2 (1.5 puntos). En un sistema monoprocesador llegan a procesarse cuatro tareas con los siguientes patrones de ejecución:

Proceso	Llegada	CPU	E/S	CPU
P1	2	1	2	4
P2	0	5	3	3
P3	3	2	3	2
P4	4	1	6	2

a. (**1,5 puntos**) Use el diagrama que aparece a continuación para simular la ejecución de los procesos indicados en la tabla anterior, considerando un sistema monoprocesador y una política de planificación de dos niveles con realimentación. El primer nivel, el de mayor prioridad, usa una política Round Robin con quantos de 2 unidades, mientras que el segundo nivel usa una política FCFS. Al principio todos los procesos estarán en la cola de máxima prioridad y los procesos finalizarán una vez finalice su patrón de ejecución. Cuando un proceso agota su quanto será penalizado y enviado a la cola de nivel inferior. Siempre que un proceso vuelve de una operación E/S, entrará directamente en la cola de mayor prioridad.

Muestre en cada momento si un proceso está siendo ejecutado, está bloqueado o listo para ejecutar. Adicionalmente, incluya los procesos que están en Cola y ordénelos en orden de ejecución. Emplee la siguiente notación para representar los estados de los procesos:

- En ejecución: marcar con "**X1**" cuando la tarea esté en nivel 1 y con "**X2**" cuando esté en nivel 2.
- Bloqueado por E/S: marcar con "**O**"
- Listo para ejecutar: marcar con "**--**"

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P1																					
P2																					
P3																					
P4																					
Cola L1																					
Cola L2																					

b. (**0.5 puntos**) Calcule los tiempos de espera y ejecución de cada tarea así como porcentaje de uso de la CPU.

Tarea	**Tiempo de Espera**	**Tiempo de Ejecución**
P1		
P2		
P3		
P4		

% uso de CPU:

Cuestión 3 (2 puntos). Supóngase que existe un puente en el que sólo pueden circular coches en un sentido, existiendo coches que quieren cruzar el puente en ambos sentidos. Se desea simular con hilos POSIX la circulación de varios coches que quieren cruzar este puente, cada uno en un sentido (0: de izquierda a derecha, 1: de derecha a izquierda). El esquema general de código para los coches sería:

```
void car(int dir)
{
	enter_bridge(dir);
	cross_bridge(dir);
	exit_bridge(dir);
}
```

Las funciones `enter_bridge` y `exit_bridge` permiten a los hilos sincronizarse para cruzar el puente (ejecución de la función `cross_bridge`) cumpliendo las siguientes restricciones:

- Los coches deben comenzar a cruzar en el orden de llegada al puente (orden global común a los dos sentidos).
- El puente es de un sólo carril por lo que los coches deben esperar si hay coches cruzando en sentido contrario.
- El puente no es muy robusto por lo que los coches deben esperar si ya hay 4 coches cruzando.

Implemente las funciones `enter_bridge` y `exit_bridge`, añadiendo las variables globales necesarias y los mecanismos de sincronización apropiados. Para la resolución del problema se podrán emplear tanto semáforos, como mutex y variables de condición, será elección del alumno. No es necesario incluir el código del programa `main()` que crea los hilos e inicializa los recursos de sincronización, pero si se utilizan semáforos es necesario indicar el valor inicial que se les daría.

Cuestión 4. (2 puntos) En un sistema GNU/Linux se ejecuta el siguiente programa que simula un sistema de procesamiento de tareas utilizando procesos e hilos. Se van escribiendo mensajes en un fichero de registro. El propósito del fichero es determinar si se han ejecutado o no todas las tareas:

```
#define MAX_TASKS 5
int fd;

void* task_handler(void* arg) {
    int task_id = *(int*)arg;
    char buffer[32];

    sprintf(buffer, "Task %d started\n", task_id);
    write(fd, buffer, strlen(buffer));
    // Simulate task processing time
    sleep(task_id + 1);
    sprintf(buffer, "Task %d finished\n", task_id);
    write(fd, buffer, strlen(buffer));
    return NULL;
}

int main(void) {
    pthread_t tasks[MAX_TASKS];
    int task_ids[MAX_TASKS];
    char buffer[32];
    pid_t pid;
    int i, status;

    for (i = 0; i < MAX_TASKS; i++) {
         pid = fork();
         fd = open("tasks_log.txt", O_CREAT | O_WRONLY, 0644);
         if (pid == 0) { // Child process
              sprintf(buffer, "Child %d gets the task\n", i);
              write(fd, buffer, strlen(buffer));
              task_ids[i] = i;
              pthread_create(&tasks[i], NULL, task_handler, (void*)&task_ids[i]);
              pthread_join(tasks[i], NULL);
              close(fd);
              exit(EXIT_SUCCESS);
         } else {
              sprintf(buffer, "Process delegates in %d\n", i);
              write(fd, buffer, strlen(buffer));
         }
    }

    while ((pid = wait(&status)) > 0);
    close(fd);
    return 0;
}
```

Responda razonadamente las siguientes preguntas:

a. Indique cuántos procesos e hilos se crearán al ejecutar el programa. ¿Cuántos procesos/hilos se ejecutarán de forma concurrente como máximo?

b. Dibuje el grafo de ejecución de los procesos e hilos. Justifique si es posible determinar o no el orden exacto en el que se ejecutarán los procesos e hilos creados.

c. ¿Quedarán correctamente registrados todos los mensajes en el fichero tasks_log.txt? Suponga que de partida el fichero no existe. Justifique su respuesta describiendo un posible contenido del fichero.

d. Proponga una modificación sencilla en el código para gestionar correctamente el descriptor de fichero y que todos los mensajes se registren en tasks_log.txt.

Cuestión 5 (2 puntos) En un sistema tipo UNIX, con memoria virtual con paginación bajo demanda, se compila y enlaza estáticamente el siguiente código.

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

#define N 7+1 // N+1
#define K 4

int factorial = 1;
int *buffer;

void *binomial_divisor(void *arg)
{
    int index = N-K;
    int divisor = buffer[K] * buffer[index];
    // D
    printf("Binomial divisor is: %d\n", divisor);
    return NULL;
}

int main(int argn, char *argv[])
{
    pid_t pid;
    int i;
    pthread_t tid;
    // A
    int shd = shm_open("BUFFER", O_CREAT|O_RDWR, 0777);
    ftruncate(shd, N * sizeof(int));
    buffer = (int*)mmap(NULL, N*sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED, shd, 0);

    if ((pid = fork()) == -1) {
       perror("fork");
       exit(EXIT_FAILURE);
    }
    // B
    if (pid == 0) {
        for (i = 0; i < N; i++) {
            if (i == 0){
                buffer[i] = factorial;
            } else{
                buffer[i] = factorial * i;
                factorial = factorial * i;
            }
            printf("Factorial %d\n",factorial);
        }
        exit(1);
    }
    // C
    while (wait(NULL) != -1);
    printf("Binomial dividend is: %d\n", buffer[N-1]);
    pthread_create(&tid, NULL, binomial_divisor, NULL);
    pthread_join(tid, NULL);
    munmap(buffer, N*sizeof(int));
    return 0;
}
```

Responda a las siguientes preguntas:

a) (**0.5 puntos**) Complete la siguiente tabla, indicando para cada símbolo la región de memoria en la que aparece en el mapa de memoria de ambos procesos, la protección de dicha región y si tiene soporte en el fichero ejecutable o no lo tiene:

Símbolo	*Región de Memoria Padre*	*Región de Memoria Hijo*	*Protección (RWX)*	*Soporte en Ejecutable (si/no)*
binomial_divisor				
factorial				
buffer				
divisor				

b) (**0.5 puntos**) Indique las regiones de memoria que existen en los espacios de direcciones de los procesos en los puntos marcados en el código con los comentarios // A y // B.

c) (**0.5 puntos**) Describa los cambios que se producen en los mapas de memoria de los dos procesos desde la línea marcada con el comentario // B hasta: a) la línea marcada con el comentario // C y b) la línea marcada con el comentario // D. Justifique el valor que tomará la variable divisor en el proceso padre.

d) (**0.5 puntos**) Justifica cuántos fallos de página se producen desde el punto marcado con el comentario // A al punto marcado con el comentario // C, indicando la línea en la que se produce cada fallo y la región de memoria a la que corresponde el marco de página asignado.