NOMBRE Y APELLIDOS: _____ NIA: _____

Examen de Sistemas Operativos - 1er Parcial
21 de Enero de 2010

NOTAS:

* Para la realización del presente examen se dispondrá de 2 horas.
* No se pueden utilizar libros ni apuntes, ni usar móvil (o similar).
* Responda cada pregunta en hojas distintas.

Teoría 1. Explique en detalle cómo se pasa una llamada al sistema operativo.

Teoría 2. Explique las diferencias, desde el punto de vista de imagen de memoria, de un proceso sin threads y con threads.

NOMBRE Y APELLIDOS: _____ NIA: _____

Examen de Sistemas Operativos - 1er Parcial
21 de Enero de 2010

Ejercicio 1. Dado el programa que se muestra a continuación, responda a las cuestiones:

```
#include <stdio.h>
#include <stdlib.h>
main() {
   int pid,i, m=10;
   int tiempoinicial, tiempoactual;
   tiempoinicial = time(NULL);//time devuelve el tiempo actual en segundos
   tiempoactual = time(NULL) - tiempoinicial;
   printf("%d:Inicio del programa \n",tiempoactual );
   for(i=0; i<3; i++) {
      pid=fork();
      sleep(1);
      switch(pid) {
         case -1:
            perror("Error en la creación de procesos");
            exit(-1);
         case 0:
            m++;
            tiempoactual = time(NULL) - tiempoinicial;
            printf("%d:Hijo %d m=%d\n",tiempoactual, i, m);
            sleep(2);
            exit(0);
         default:
            tiempoactual = time(NULL) - tiempoinicial;
            printf("%d:Creado el proceso %d\n", tiempoactual, i);
            if( i%2 == 0 ) {
               wait(NULL); //wait espera que finalice un hijo cualquiera
               tiempoactual = time(NULL) - tiempoinicial;
               printf("%d:Finalizó un proceso, valor de m=%d\n",
                           tiempoactual,m);
            } //fin if
      } //fin switch
   } //fin for
   wait(NULL);
   tiempoactual = time(NULL) - tiempoinicial;
   printf("%d:Finalizó un proceso, valor de m=%d",tiempoactual, m);
} //fin main
```

a) Escribir los mensajes que se escriben por pantalla y en qué instante, suponiendo que el mensaje de Inicio del programa aparece en el instante 0.

b) ¿Cuántas variables m se crean en memoria?

NOMBRE Y APELLIDOS: _____ NIA: _____

Examen de Sistemas Operativos - 1er Parcial
21 de Enero de 2010

Ejercicio 2. Dado el siguiente programa, y considerando que cada vez que un proceso tiene el procesador ejecuta la instrucción de suma (var++) 100.000.000 de veces por segundo, completar la tabla proporcionada en los siguientes casos:

a) Planificación con prioridades expulsiva.
b) Round Robin con rodaja de tiempo de 2 seg, expulsiva.

```
int var=0; //variable global
int fin=0; //variable global

void terminar(void) {
    // cuando se termina un proceso se imprime el valor de var
    printf("Proceso %d: var=%d\n", getpid(), var);
    fin = 1;
}

main() {
    int pid,i;
    signal(SIGALRM, terminar); /* se establece la acción a ejecutar cuando se
                                  reciba una alarma. Esta acción es la función
                                  terminar() */
    for(i=0; i<3; i++) {
        sleep(1);
        pid=fork();
        switch(pid) {
            case -1:
                perror("Error en la creación de procesos");
                exit(-1);
            case 0:
                alarm(5);  /* Se ejecuta siempre de inmediato al crear el hijo
                              porque cada proceso que se crea tiene inicialmente la
                              misma prioridad que el padre (prioridad = P) */
                nice(i%2 + 1); /*Disminuye la prioridad del proceso tantas
                                 unidades como se indique en el parámetro*/
                while(!fin)
                    var++; /* Se ejecuta la iteración hasta que fin cambia de
                              valor, es decir, cuando se recibe la alarma y se
                              trata en la función terminar */
              exit(0);
        } //fin switch
   } //fin for
} fin main
```

TABLA A COMPLETAR

PROCESO	INSTANTE DE LLEGADA	PRIORIDAD	INSTANTE FINAL DE EJECUCIÓN	VALOR DE VAR IMPRESO -Apartado a) -	VALOR DE VAR IMPRESO -Apartado b) -
Padre	0	P	3	no imprime var	no imprime var
Hijo 0	1	P - 1	6		
Hijo 1				0	
Hijo 2	3				

NOMBRE Y APELLIDOS: _____ NIA: _____

Examen de Sistemas Operativos - 1er Parcial
21 de Enero de 2010

Ejercicio 3.

a) Escribir en código C una función llamada mi_cat que lea el contenido de un fichero (cuyo nombre se recibe por el primer parámetro) y lo escriba en un descriptor recibido por el segundo parámetro. Ayudarse del siguiente código:

```
void mi_cat(char *nombre_fichero, int fd_salida){

        int fd_entrada = open(nombre_fichero, O_RDONLY);

        if( fd_entrada < 0 ) {
            perror(Error al abrir el fichero);
            exit(-1);
        }

        /* RELLENAR POR EL ALUMNO */
            // LEER DE fd_entrada Y ESCRIBIR EN fd_salida
        /* FIN RELLENAR POR EL ALUMNO */

        if( close(fd_entrada) < 0 ) {
            perror(Error al cerrar el fichero);
            exit(-1);
        }
}
```

b) Escribir un programa en código C que ejecute:

mi_cat fichero_alumnos.txt | grep manuel

El proceso que ejecute mi_cat debe enviar el contenido del archivo fichero_alumnos.txt por la tubería, y el proceso que ejecute grep coge los datos de la tubería y filtra sólo las líneas en las que aparece la palabra manuel.

- Se deben utilizar procesos pesados comunicándolos mediante tuberías o pipes.
- El comando mi_cat debe ejecutarse haciendo uso de la función del apartado a), en lugar de utilizar execvp para ejecutarlo. En argv[1][0] se encuentra el nombre del fichero fichero_alumnos.xt.
- Para el comando grep suponer que en argv[2] se encuentra el comando con sus parámetros. De tal forma que para ejecutar este mandato grep, se debe utilizar la llamada al sistema execvp, de la forma: `execvp(argv[2][0], argv[2]);`

```
int main(int argc, char **argv[]){
      /* RELLENAR POR EL ALUMNO */
}
```

NOMBRE Y APELLIDOS: _____ NIA: _____

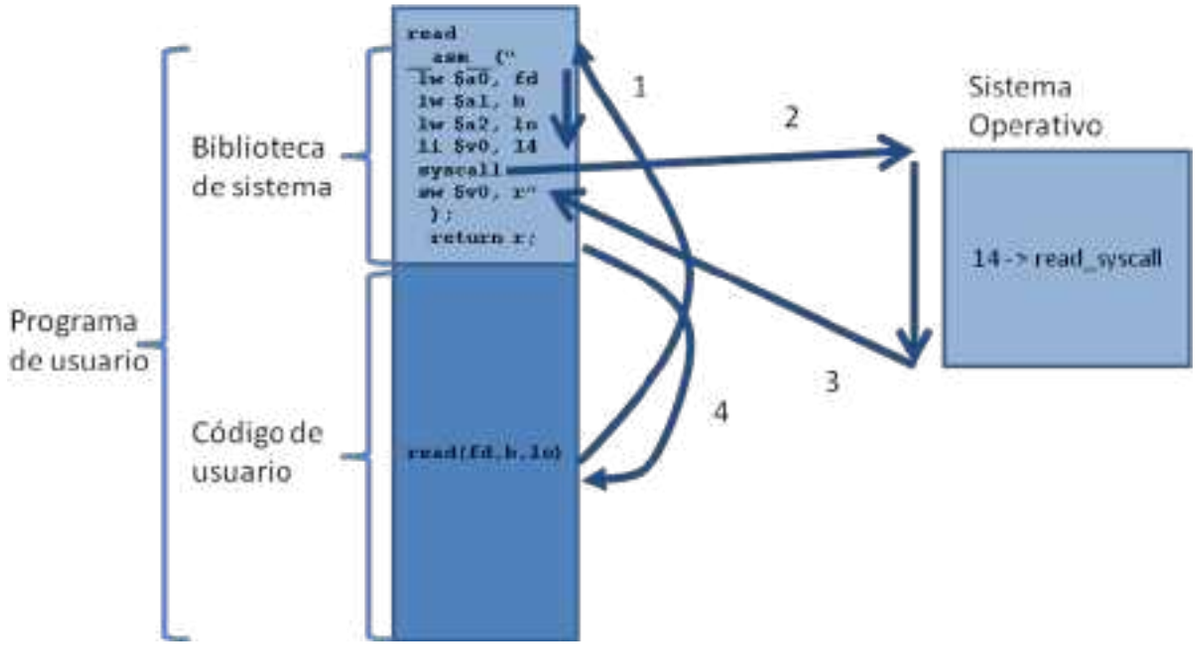
Examen de Sistemas Operativos - 1er Parcial
21 de Enero de 2010

SOLUCIÓN

Teoría 1.

El sistema operativo se activa cuando debe responder a una petición de servicio de un proceso, es decir una llamada al sistema. Esta petición se realiza mediante la llamada a una biblioteca que incluye funciones de interfaz de programación (API). Cada una de ellas se corresponde con algún servicio del sistema operativo. La función es un envoltorio para el código que invoca el servicio del sistema operativo y la proporciona normalmente el fabricante del sistema operativo. Puesto que esta función incluye la ejecución de una instrucción de trap que transfiere el control al sistema operativo mediante la generación de una interrupción y eleva el nivel de ejecución de la CPU para pasar a modo privilegiado.

A continuación se incluye un esquema con el proceso de una llamada a sistema:



Como se puede ver, el usuario ejecuta la rutina de API de llamada ala sistema read, que llama a la biblioteca del sistema (1). Esta prepara los argumentos, carga el código de llamada al sistema (read_syscall, 14) y ejecuta un trap mediante la instrucción syscall (2). Este trap invoca al sistema operativo que ejecuta read_sycall al ver el código 14 y devuelve el resultado en V0 (3). La rutina de biblioteca retorna a la de usuario (4) y devuelve el resultado de la llamada al sistema.

NOMBRE Y APELLIDOS: _____ NIA: _____

Examen de Sistemas Operativos - 1er Parcial
21 de Enero de 2010

Teoría 2.

Un proceso sin threads incluye un único hilo de ejecución, por tanto su imagen de memoria incluye un número de regiones variables, entre las cuáles necesita, como mínimo, una región de texto, una de datos dinámicos y una pila.

Un proceso con threads tiene varios flujos de ejecución que comparten casi toda la imagen de memoria. Pero la imagen de memoria debe tener una parte específica de cada thread para dar soporte a dicha ejecución y esa parte no se comparte.

La información de memoria específica de cada thread comprende:

Identificador de thread (tid)

Contador de programa (PC)

Conjunto de registros (registers)

Pila (Stack)

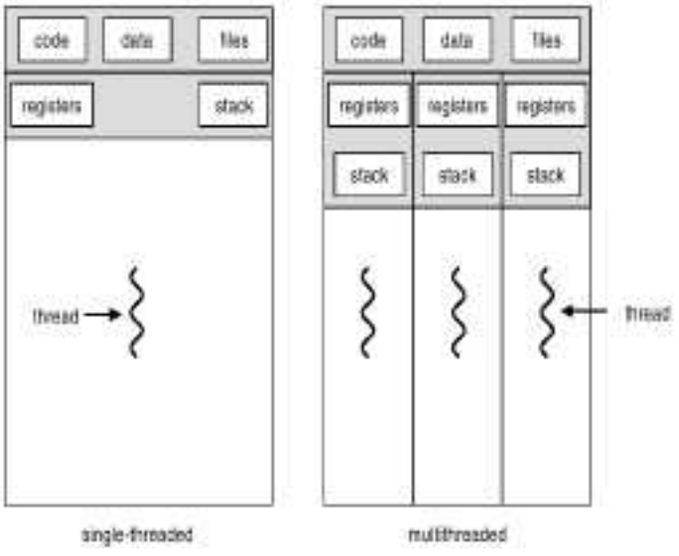
La información de memoria compartida por todos los threads de un proceso es:

Mapa de memoria (sección de código, sección de datos, shmem)

Ficheros abiertos

Señales, semáforos y temporizadores.

La figura siguiente muestra ambos casos:



NOMBRE Y APELLIDOS: _____ NIA: _____

Examen de Sistemas Operativos - 1er Parcial
21 de Enero de 2010

Ejercicio 1:

```
0:Inicio del programa
1:Hijo 0 m=11
1:Creado el proceso 0
3:Ha finalizado un proceso
4:Hijo 1 m=11
4:Creado el proceso 1
5:Hijo 2 m=11
5:Creado el proceso 2
6:Ha finalizado un proceso
7:Ha finalizado un proceso
```

Ejercicio 2:

PROCESO	INSTANTE DE LLEGADA	PRIORIDAD	INSTANTE FINAL DE EJECUCIÓN	VALOR DE VAR IMPRESO -Apartado a) -	VALOR DE VAR IMPRESO -Apartado b) -
Padre	0	P	3	no imprime var	no imprime var
Hijo 0	1	P - 1	6	500.000.000	300.000.000
Hijo 1	2	P - 2	7	0	0
Hijo 2	3	P - 1	8	200.000.000	400.000.000

En el caso B también se puede dar como admisible que el valor impreso de var por el hijo 0 sea 400.000.000 y el del hijo 2 sea 300.000.000.

NOMBRE Y APELLIDOS: _____ NIA: _____

Examen de Sistemas Operativos - 1er Parcial
21 de Enero de 2010

Ejercicio 3:

a)

```
void mi_cat(char *nombre_fichero, int fd_salida){

    int fd_entrada = open(nombre_fichero, O_RDONLY);

    if( fd_entrada < 0 ) {
        perror(Error al abrir el fichero);
        exit(-1);
    }

    /* RELLENAR POR EL ALUMNO */

    int n;
    char buffer[512];
    memset(buffer, 0, 512);
    while( (n = read(fd_entrada, buffer, 512)) > 0){
        if( write(fd_salida, buffer, n) < 0) {
            perror(Error al escribir en el descriptor de salida);
            exit(-1);
        }
        memset(buffer, 0, 512); //limpio la memoria para la lectura
    }

    /* FIN RELLENAR POR EL ALUMNO */

    if( close(fd_entrada) < 0 ) {
        perror(Error al cerrar el fichero);
        exit(-1);
    }
}
```

NOMBRE Y APELLIDOS: _____ NIA: _____

Examen de Sistemas Operativos - 1er Parcial
21 de Enero de 2010

b) Solución suponiendo que el hijo ejecuta mi_cat y que el padre ejecuta grep.

```
int main(int argc, char **argv[]){
    int pid;
    int tuberia[2];
    if( pipe(tuberia) < 0) {
        perror(Error al crear la tuberia);
        exit(-1);
    }
    pid=fork(); //se crea el proceso hijo
    switch(pid) {
        case -1:
            perror(Error en la creación de procesos);
            exit(-1);
        case 0:
            mi_cat(argv[1][0], tuberia[1]);
            exit(0);
        default:
            wait(NULL); //espero a que termine el hijo
            close(0);
            dup(tuberia[0]);
            close(tuberia[0]);
            close(tuberia[1]); //el padre no lo utiliza
            if( execvp(argv[2][0], argv[2]) < 0)
                exit(-1);
            break;
    } //fin switch
}
```

Solución suponiendo que el padre ejecuta mi_cat y que el hijo ejecuta grep.

```
int main(int argc, char **argv[]){
    int pid;
    int tuberia[2];
    if( pipe(tuberia) < 0) {
        perror(Error al crear la tuberia);
        exit(-1);
    }
    pid=fork(); //se crea el proceso hijo
    switch(pid) {
        case -1:
            perror(Error en la creación de procesos);
            exit(-1);
        case 0:
            close(0);
            dup(tuberia[0]);
            close(tuberia[0]);
            close(tuberia[1]); //el hijo no lo utiliza
            if( execvp(argv[2][0], argv[2]) < 0)
                exit(-1);
        default:
```

NOMBRE Y APELLIDOS: _____ NIA: _____

Examen de Sistemas Operativos - 1er Parcial
21 de Enero de 2010

```
            mi_cat(argv[1][0], tuberia[1]);
            close(tuberia[1]); //el padre ya no lo necesita
            close(tuberia[0]); //el padre no lo utiliza
            wait(NULL);
            break;
    } //fin switch
}
```