Material permitido: **Calculadora NO programable.** Tiempo: **2 horas.** N1	**Aviso 1:** Todas las respuestas deben estar razonadas. **Aviso 2:** Escriba sus respuestas con una letra **lo más clara posible.** **Aviso 3: No use *Tipp-ex*** o similares (atasca el escáner).

ESTE EXAMEN CONSTA DE 5 PREGUNTAS

1. Responda a las siguientes cuestiones:

a) *(0.5 p)* Explique el funcionamiento del siguiente comando:

```
grep patrón fichero1 fichero2 … ficheroN
```

b) *(0.5 p)* Explique el funcionamiento de la siguiente llamada al sistema y el significado de sus parámetros:

```
resultado = mount(dispositivo,dir,flags);
```

RESPUESTA:

a) V. Apartado. 2.9) El comando `grep` busca una cadena de texto definida en "`patrón`" dentro dentro de uno o varios ficheros `fichero1 fichero2 … ficheroN`, imprimiendo por la salida estándar la línea en la que encuentra la coincidencia.

b) (V. Apartado 8.4.2) La llamada al sistema `mount` permite montar un sistema de ficheros desde un programa. Su sintaxis, en los sistemas UNIX clásicos, es:

```
resultado = mount(dispositivo,dir,flags);
```

Donde `dispositivo` es la ruta de acceso del fichero del dispositivo del disco en el cuál se encuentra el sistema de ficheros que se va a montar, `dir` es la ruta de acceso del directorio sobre el que se va a montar el sistema de ficheros y `flags` es una máscara de bits que permite especificar diferentes opciones. En concreto el bit menos significativo de `flags` se utiliza para revisar los accesos de escritura sobre el sistema de ficheros. Si vale 1, la escritura estará prohibida, por lo que sólo se podrán hacer accesos de lectura; en caso contrario, la escritura estará permitida, pero de acuerdo a los permisos individuales de cada fichero.

Si la llamada se ejecuta con éxito en `resultado` se almacena el valor 0. En caso contrario, se almacena el valor -1.

2. *(2 p)* Describa las acciones que realiza el núcleo cuando un proceso realiza la llamada al sistema `shmget`.

RESPUESTA:

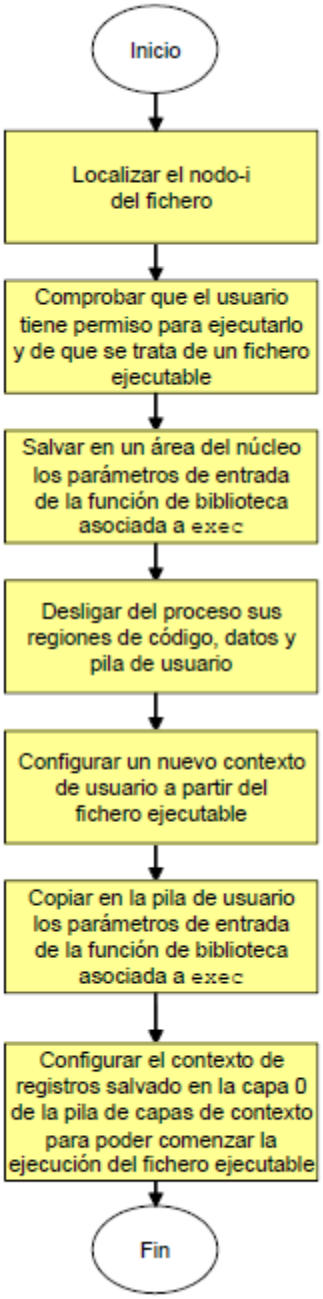
(V. Apartado. 6.1.1.3*).* Cuando un proceso realiza la llamada al sistema `shmget` el núcleo realiza las siguientes acciones:

- Busca en la *tabla de memoria compartida* la región asociada con el parámetro `key`, si encuentra dicha región y el proceso tiene los permisos de acceso correctos entonces devuelve el descriptor `shmid`.
- Si no encuentra la región asociada con el parámetro `key` y el usuario ha configurado el indicador `IPC_CREAT` de `flags` para crear una nueva región entonces:

- Comprueba que el tamaño especificado `size` se encuentra entre los límites mínimo y máximo permitidos
- Asigna una región mediante el uso del algoritmo `allocreg()`.
- Salva los permisos, tamaño y un puntero a la *tabla de regiones* dentro de la estructura `shmid_ds` asociada a la entrada `shmid` de la *tabla de memoria compartida*.
- Activa un bit en la entrada de la *tabla de regiones* asignada a la región de memoria compartida `shmid` para identificarla como una región de memoria compartida.
- También en la entrada de la *tabla de regiones* asignada a la región de memoria compartida `shmid` el núcleo activa un bit para indicar que dicha región no debe ser liberada cuando el último proceso que la comparta termine. Por lo tanto, los datos en una región de memoria compartida permanecerán intactos incluso aunque ningún proceso comparta ya dicha región

3. (*2 p*) Dibuje un diagrama, **adecuadamente rotulado**, que esquematice las principales acciones que realiza el núcleo durante la ejecución del algoritmo `exec()`.

RESPUESTA:
(V. Apartado 4.8.2)



4. *(2p)* Escriba un programa en C que realice la creación de un conjunto de 3 semáforos asociados a la llave creada a partir del fichero “esllave” y la clave ‘J’. El conjunto de semáforos se crea con permisos de lectura y modificación para el usuario.

RESPUESTA:
(V. Apartado 6.4.2)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
...
int semid;
key_t llave;
...
llave=ftok("esllave", 'J');
if(llave==(key_t)-1)
{
/*Se ha producido un error al crear la llave.
Código de tratamiento del error*/
}
/*Creación del conjunto de semáforos*/
semid=semget(esllave, 3, IPC_CREAT| 0600);
if (semid==-1)
{
/*Error al crear el conjunto de semáforos.
Código de tratamiento del error*/
}
```

5. *(3 p)* Conteste razonadamente a los siguientes apartados:

a) (1.5 p) Explicar el significado de las sentencias enumeradas (**[ ]**) del programa que se muestra en la página siguiente.

b) *(1.5 p)* Explique el funcionamiento del programa explicando detalladamente su salida asumiendo que no hubiese otros programas que compitan con él por el uso de CPU.

La pregunta continua en la página siguiente

```
        #include <signal.h>
        #include <stdio.h>
        #include <stdlib.h>
        #include <unistd.h>
        #include <time.h>
        #include <sys/times.h>

        void despertar(int sig);
        void continuar(int sig);
        void acabar(int sig);

        void main(void)
        {
             int pid;
[1]          signal(SIGCHLD,despertar);
             signal(SIGCONT,continuar);
             signal(SIGTERM,acabar);
[2]          if ((pid=fork())==-1){
                  perror("Error en fork:");
                  exit(1);
             }
             if (pid!=0) {
[3]               pause();
                  printf("del examen DyASO ");
[4]               while(1);
                  printf("erroneo.\n");
[5]               exit(0);
             }
             else{
[6]               sleep(1);
                  printf("Ejercicio 5 ");
[7]               kill(getppid(),SIGCONT);
                  sleep(1);
             }
        }

        void despertar(int sig)
        {
        struct tms T;
        time_t cpu;
[8]     times(&T);
        printf("correcto");
        cpu=T.tms_utime + T.tms_stime + T.tms_cutime + T.tms_cstime;
        printf(" tras %ld ms de ejecución activa.\n",
                                            (1000*cpu)/sysconf(_SC_CLK_TCK));
[9]     raise(SIGTERM);
        }

        void continuar(int sig) {}

        void acabar(int sig) {exit(0);}
```

RESPUESTA
Apartado A:

[1] La llamada al sistema `signal(SIGCHLD,despertar);` instala la función "despertar" como manejador de la señal `SIGCHLD`. Esto significa que cada vez que el proceso reciba la señal `SIGCHLD` (terminación de uno de sus hijos) ejecutará la función `despertar`.

[2] `fork()` es una llamada al sistema cuyo efecto es crear un proceso hijo del proceso actual duplicando el proceso inicial o padre. `fork()` devuelve al proceso padre el pid del proceso hijo y devuelve 0 al proceso hijo, dicho valor de retorno se almacena en la variable `pid`. En caso de error al crear el proceso nuevo se ejecutaría el bloque de código correspondiente al `if` (esto es, se mostraría un mensaje de error y finalizaría el programa).

[3] `pause()` es una llamada al sistema utilizada por el proceso padre para esperar hasta la terminación de uno de sus hijos antes de continuar su ejecución.

[4] `while(1);` `realiza` una "espera activa", el proceso quedará atrapado en un bucle infinito hasta que sea interrumpido (ya que la condición del while siempre se cumple). A diferencia de `wait()` que mantiene el proceso dormido, la espera activa mantiene el procesador trabajando.

[5] `exit()` es la llamada al sistema que finaliza un proceso devolviendo un código de retorno. Por convenio el código de retorno de un programa que finaliza con éxito es 0 mientras que otros códigos suelen indicar un error.

[6] `sleep(1)` es una llamada al sistema que pone a dormir al proceso que la invoca durante 1 segundo.

[7] `kill` es una llamada al sistema que permite enviar una señal a un determinado proceso, mientras que `getppid()` es una llamada al sistema que devuelve el pid del proceso padre, de este modo `kill(getppid(),SIGCONT)` envía al proceso padre del proceso que la invoca la señal `SIGCONT`.

[8] La llamada al sistema `times(&T)` escribe en la estructura de tipo tms, T los tiempos de ejecución del proceso en curso, en particular sus campos son `tms_utime`, `tms_stime`, (tiempo de ejecución en modo usuario y sistema respectivamente) T.tms_cutime y `T.tms_cstime` (tiempo de ejecución en modo usuario y sistema consumido por sus descendientes).

[9] `raise(SIGTERM)` es una llamada al sistema que permite a un proceso enviarse una señal a si mismo. En este caso se envía la señal `SIGTERM` que se usa para terminar un proceso.

Apartado B:
El funcionamiento del programa es el siguiente:

En primer lugar el programa asocia las señales `SIGCHLD`, `SIGCONT` y `SIGTERM` con las funciones `despertar`, `continuar` y `acabar`.

A continuación el proceso se bifurca dando lugar a un padre y a un hijo, si se produjese algún error el programa imprimiría `Error en fork:` seguido de la descripción del error y terminaría con la condición de error 1.

El proceso padre ejecutará el bloque de código siguiente a `if(pid!=0)` y el proceso hijo ejecutará el bloque de código siguiente al `else`.

El proceso padre, en primer lugar, hace una pausa a la espera de recibir una señal, por lo que comenzará a ejecutarse el hijo.

El proceso hijo se dormirá durante un segundo de tal modo que durante ese tiempo tanto el padre como el hijo permanecerán dormidos. Tras despertarse imprimirá "`Ejercicio 5 `" por la salida estándar, enviará una señal `SIGCONT` a su padre y volverá a dormirse otro segundo.

El padre, al recibir la señal `SIGCONT`, ejecutará la acción continuar (que no realiza ninguna acción) y a continuación se despertará. Aunque la función continuar no tenga código, su existencia evita que el padre ignore la señal.

El padre imprime por salida estándar "`del examen DyASO `" y luego entra en un bucle infinito `while(1)` haciendo lo que se denomina una espera activa ya que el procesador está ejecutando las instrucción de comprobación del bucle continuamente.

Al pasar un segundo, el hijo se despertará y finalizará su ejecución. Al hacerlo el Sistema Operativo envía automáticamente al padre una señal `SIGCHLD`, señal que el padre no ignora. De este modo, cuando el padre vuelva a ejecutarse llamará a la función `despertar`.

Dicha función calculará el tiempo que el proceso y su hijo han utilizado la cpu. Convierte dicho número en milisegundos y muestra por pantalla "`tras`" seguido del número de milisegundos de ejecución. y a continuación "`ms de ejecución activa.`" Y un salto de línea.

Puesto que se asume que no hay más procesos que compiten por usar la cpu, en el momento en que se duerme el proceso hijo se planifica inmediatamente el padre y por lo tanto el `while` se estará ejecutando hasta que el hijo despierte (un segundo aproximadamente).

A continuación, la función `despertar` enviará al proceso padre la señal `SIGTERM` (recordemos que se la señal se ejecuta en el contexto del proceso padre que es quien la ha recibido) .

El proceso padre al recibir la señal `SIGTERM` llama a la función `acabar` que finaliza el proceso. De este modo la línea `printf("erroneo.\n")` nunca llega a ejecutarse en el proceso padre.

De este modo tras dos segundos la salida será similar a la siguiente:

Ejercicio 5 del examen DyASO correcto tras 1000 ms de ejecución activa.

NOTA: Sea cual sea el orden de planificación de los procesos padre e hijo el resultado es siempre el mismo ya que el proceso padre se queda a la espera de señales por parte del hijo y eso sincroniza las acciones del padre y del hijo.