Makefile (0,5 puntos)

Crea un fichero Makefile que permita generar todos los programas del enunciado a la vez, así como también cada uno por separado. Añade una regla (clean) para borrar todos los binarios y/o ficheros objeto y dejar sólo los ficheros fuente. Los programas se han de generar si, y solo si, han habido cambios en los ficheros fuente.

Control de errores (0,5 puntos)

Para todos los programas que se piden a continuación hace falta comprobar los errores de TODAS las llamadas a sistema (excepto el write por pantalla), controlar los argumentos de entrada y definir la función Usage().

Ejercicio 1 (4 puntos)

Haz un programa llamado **watchPID.c** para monitorizar la ejecución de un proceso. Este programa puede recibir dos o tres argumentos. El primero, *interval*, será un número con valor entre 1 y 5 (ambos inclusive) que corresponderá con el intervalo de monitorización en segundos (es decir, cada cuanto tiempo se monitoriza el proceso). El segundo, *count*, será un número con valor entre 1 y 10 (ambos inclusive) que corresponderá con el número de veces que se ha de monitorizar el proceso. El tercero, *pid*, será un identificador de proceso. Este parámetro es opcional, y si no se especifica se tendrá que monitorizar el proceso actual (es decir, el proceso padre inicial). Después de comprobar los parámetros, el programa *watchPID* creará, de manera concurrente, tantos procesos como indique el parámetro *count*. A continuación, el programa ha de esperar a los procesos hijo (pero sin recoger su código de finalización) y acabar. Cada proceso hijo se encargará de hacer la monitorización de un intervalo determinado usando el comando `top`. En concreto, cada hijo tiene que escribir por pantalla un mensaje con su índice, en función de su orden de creación, y con su PID (ver el ejemplo de la salida), y ha de mutar a "`top -b -n 1 -p <pid>`", donde "<pid>" representa el PID del proceso a monitorizar. Para configurar el intervalo, antes de mutar, todos los hijos, excepto el primero, han de configurar una alarma con una duración resultante de multiplicar el parámetro *interval* por el índice del proceso, y esperar que salte esta alarma sin consumir CPU.

Ejemplo de la salida:

```
$ ./watchPID 1 2
INDEX: 0, PID: 5382
top - 15:48:43 up 30 min,  1 user,  load average: 0,06, 0,20, 0,17
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,0 sy,  0,0 ni,100,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem :   1983,2 total,    482,0 free,    420,6 used,   1080,6 buff/cache
MiB Swap:      0,0 total,      0,0 free,      0,0 used.   1391,0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   5381 root      20   0    2356    572    508 S   0,0   0,0   0:00.00 watchPID

INDEX: 1, PID: 5383
top - 15:48:44 up 30 min,  1 user,  load average: 0,05, 0,20, 0,17
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,0 us,  3,2 sy,  0,0 ni, 96,8 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem :   1983,2 total,    482,3 free,    420,4 used,   1080,6 buff/cache
MiB Swap:      0,0 total,      0,0 free,      0,0 used.   1391,3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   5381 root      20   0    2356    572    508 S   0,0   0,0   0:00.00 watchPID
```

Ejercicio 2 (4,5 puntos)

Copia **watchPID.c** creando un nuevo fichero llamado **watchPID2.c** y haz las siguientes modificaciones:

- El proceso padre, antes de crear ningún proceso, tiene que bloquear los signals SIGUSR1 y SIGUSR2. Después de haber creado todos los hijos y antes de esperar su finalización, enviará un signal SIGUSR1 a los procesos con índice impar, y un signal SIGUSR2 a los procesos con índice par. Los valores de los PIDs de los procesos hijo, necesarios para poder enviar los signals, se tienen que guardar, durante la creación de los procesos, en un vector reservado con memoria dinámica (usando la función `malloc`). Es necesario liberar la memoria dinámica reservada antes de acabar la ejecución del programa.
- Para esperar la finalización de los procesos hijo, el padre ha de capturar el signal SIGCHLD, y mantenerse en una espera activa mientras no acaben todos los hijos. La rutina de atención al signal SIGCHLD ha de recoger el estado de finalización (proceso acabado por exit o por signal) y también el valor de finalización (código del exit o número del signal) de cada hijo y guardar esta información, ambos valores, en una estructura de datos en la posición correspondiente, según el índice de creación, de un vector reservado con memoria dinámica (usando la función `sbrk`). Es necesario liberar la memoria dinámica reservada antes de acabar la ejecución del programa. El proceso padre, una vez acabada la espera activa, escribirá la información de todos los procesos por pantalla (ver ejemplo de la salida).
- Todos los procesos hijo, antes de invocar cualquier otra operación, tienen que reprogramar el signal SIGUSR1 con una rutina de tratamiento que no haga nada y esperar la recepción de cualquier signal (sin consumir CPU).

Ejemplo de la salida:

```
$ ./watchPID2 1 4

INDEX: 1, PID: 10390
top - 16:28:00 up  1:09,  1 user,  load average: 0,08, 0,22, 0,16
Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s): 50,0 us,  0,0 sy,  0,0 ni, 50,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem :   1983,2 total,    414,7 free,    422,8 used,   1145,6 buff/cache
MiB Swap:      0,0 total,      0,0 free,      0,0 used.   1385,0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  10388 root      20   0    2492    512    444 R 100,0   0,0   0:01.14 watchPID2

INDEX: 3, PID: 10392
top - 16:28:02 up  1:09,  1 user,  load average: 0,16, 0,24, 0,17
Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s): 51,6 us,  0,0 sy,  0,0 ni, 45,2 id,  0,0 wa,  0,0 hi,  3,2 si,  0,0 st
MiB Mem :   1983,2 total,    414,7 free,    422,8 used,   1145,6 buff/cache
MiB Swap:      0,0 total,      0,0 free,      0,0 used.   1385,0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  10388 root      20   0    2492    512    444 R 100,0   0,0   0:03.14 watchPID2

SUMMARY
El proceso con índice 0 y PID 10389 ha acabado por signal 12
El proceso con índice 1 y PID 10390 ha acabado por exit con código 0
El proceso con índice 2 y PID 10391 ha acabado por signal 12
El proceso con índice 3 y PID 10392 ha acabado por exit con código 0
```

Respuestas (0,5 puntos)

Teniendo en cuenta que la rutina de tratamiento del signal SIGUSR1 no hace nada, explica en el fichero **respuesta.txt** cómo se comportaría el programa watchPID2 si modificamos la reprogramación del signal SIGUSR1 de manera que la rutina para su tratamiento sea `SIG_IGN`.

Qué hay que hacer

- El Makefile
- Los códigos de los programas en C
- La función Usage() para cada programa que sea necesario

Qué se valora

- Que sigas las especificaciones del enunciado
- Que el uso de las llamadas al sistema sea el correcto
- Que se comprueben los errores de todas las llamadas al sistema
- Que el código sea claro y correctamente indentado
- Que el Makefile tenga bien definidas las dependencias y objetivos
- Que la función Usage() muestre por pantalla como debe invocarse correctamente el programa en el caso que los argumentos recibidos no sean los adecuados
- El fichero respuesta.txt

Qué hay que entregar

Un único fichero tar.gz con el código de todos los programas, el Makefile, y el fichero respuesta.txt:

```
tar zcvf clab1.tar.gz Makefile *.c respuesta.txt
```