

Práctica 1 – Sistemas Operativos

Por Daniel Cerrato y David Garitagoitia

Ejercicio 1:

a) Para buscar en el manual la lista de funciones disponibles para el manejo de hilos se utiliza el comando “man -k pthread” y el resultado es el siguiente:

pthread_attr_destroy (3) - initialize and destroy thread attributes object
pthread_attr_getaffinity_np (3) - set/get CPU affinity attribute in thread a...
pthread_attr_getdetachstate (3) - set/get detach state attribute in thread a...
pthread_attr_getguardsize (3) - set/get guard size attribute in thread attri...
pthread_attr_getinheritsched (3) - set/get inherit-scheduler attribute in th...
pthread_attr_getschedparam (3) - set/get scheduling parameter attributes in ...
pthread_attr_getschedpolicy (3) - set/get scheduling policy attribute in thr...
pthread_attr_getscope (3) - set/get contention scope attribute in thread att...
pthread_attr_getstack (3) - set/get stack attributes in thread attributes ob...
pthread_attr_getstackaddr (3) - set/get stack address attribute in thread at...
pthread_attr_getstacksize (3) - set/get stack size attribute in thread attri...
pthread_attr_init (3) - initialize and destroy thread attributes object
pthread_attr_setaffinity_np (3) - set/get CPU affinity attribute in thread a...
pthread_attr_setdetachstate (3) - set/get detach state attribute in thread a...
pthread_attr_setguardsize (3) - set/get guard size attribute in thread attri...
pthread_attr_setinheritsched (3) - set/get inherit-scheduler attribute in th...
pthread_attr_setschedparam (3) - set/get scheduling parameter attributes in ...
pthread_attr_setschedpolicy (3) - set/get scheduling policy attribute in thr...
pthread_attr_setscope (3) - set/get contention scope attribute in thread att...
pthread_attr_setstack (3) - set/get stack attributes in thread attributes ob...
pthread_attr_setstackaddr (3) - set/get stack address attribute in thread at...
pthread_attr_setstacksize (3) - set/get stack size attribute in thread attri...
pthread_cancel (3) - send a cancellation request to a thread
pthread_cleanup_pop (3) - push and pop thread cancellation clean-up handlers
pthread_cleanup_pop_restore_np (3) - push and pop thread cancellation clean-...

pthread_cleanup_push (3) - push and pop thread cancellation clean-up handlers

pthread_cleanup_push_defer_np (3) - push and pop thread cancellation clean-u...

pthread_create (3) - create a new thread

pthread_detach (3) - detach a thread

pthread_equal (3) - compare thread IDs

pthread_exit (3) - terminate calling thread

pthread_getaffinity_np (3) - set/get CPU affinity of a thread

pthread_getattr_default_np (3) - get or set default thread-creation attributes

pthread_getattr_np (3) - get attributes of created thread

pthread_getconcurrency (3) - set/get the concurrency level

pthread_getcpuclockid (3) - retrieve ID of a thread's CPU time clock

pthread_getname_np (3) - set/get the name of a thread

pthread_getschedparam (3) - set/get scheduling policy and parameters of a th...

pthread_join (3) - join with a terminated thread

pthread_kill (3) - send a signal to a thread

pthread_kill_other_threads_np (3) - terminate all other threads in process

pthread_mutex_consistent (3) - make a robust mutex consistent

pthread_mutex_consistent_np (3) - make a robust mutex consistent

pthread_mutexattr_getpshared (3) - get/set process-shared mutex attribute

pthread_mutexattr_getrobust (3) - get and set the robustness attribute of a ...

pthread_mutexattr_getrobust_np (3) - get and set the robustness attribute of...

pthread_mutexattr_setpshared (3) - get/set process-shared mutex attribute

pthread_mutexattr_setrobust (3) - get and set the robustness attribute of a ...

pthread_mutexattr_setrobust_np (3) - get and set the robustness attribute of...

pthread_rwlockattr_getkind_np (3) - set/get the read-write lock kind of the ...

pthread_rwlockattr_setkind_np (3) - set/get the read-write lock kind of the ...

pthread_self (3) - obtain ID of the calling thread

pthread_setaffinity_np (3) - set/get CPU affinity of a thread

pthread_setattr_default_np (3) - get or set default thread-creation attributes

pthread_setcancelstate (3) - set cancelability state and type

pthread_setcanceltype (3) - set cancelability state and type

pthread_setconcurrency (3) - set/get the concurrency level

pthread_setname_np (3) - set/get the name of a thread

pthread_setschedparam (3) - set/get scheduling policy and parameters of a th...

pthread_setschedprio (3) - set scheduling priority of a thread

pthread_sigmask (3) - examine and change mask of blocked signals

pthread_sigqueue (3) - queue a signal and data to a thread

pthread_spin_destroy (3) - initialize or destroy a spin lock

pthread_spin_init (3) - initialize or destroy a spin lock

pthread_spin_lock (3) - lock and unlock a spin lock

pthread_spin_trylock (3) - lock and unlock a spin lock

pthread_spin_unlock (3) - lock and unlock a spin lock

pthread_testcancel (3) - request delivery of any pending cancellation request

pthread_timedjoin_np (3) - try to join with a terminated thread

b) Para consultar en que sección del manual están las “llamadas al sistema” escribimos el comando “man man” y observamos que están en la sección 2. Para buscar información sobre la llamada al sistema “write” usamos entonces “man 2 write”.

Ejercicio 2:

a) Para ver las líneas que contienen “molino” usamos el comando “grep ‘molino’ ‘don quijote.txt’ ” y para añadirlo al final de “aventura.txt” usamos el comando “grep ‘molino’ ‘don quijote.txt’ >> ‘aventura.txt’ ”.

b) Para conocer los archivos del directorio utilizaremos “ls” y mediante una pipeline, podremos contar el numero de líneas con “wc -l”. De esta manera nos queda el comando siguiente: “ls | wc -l”

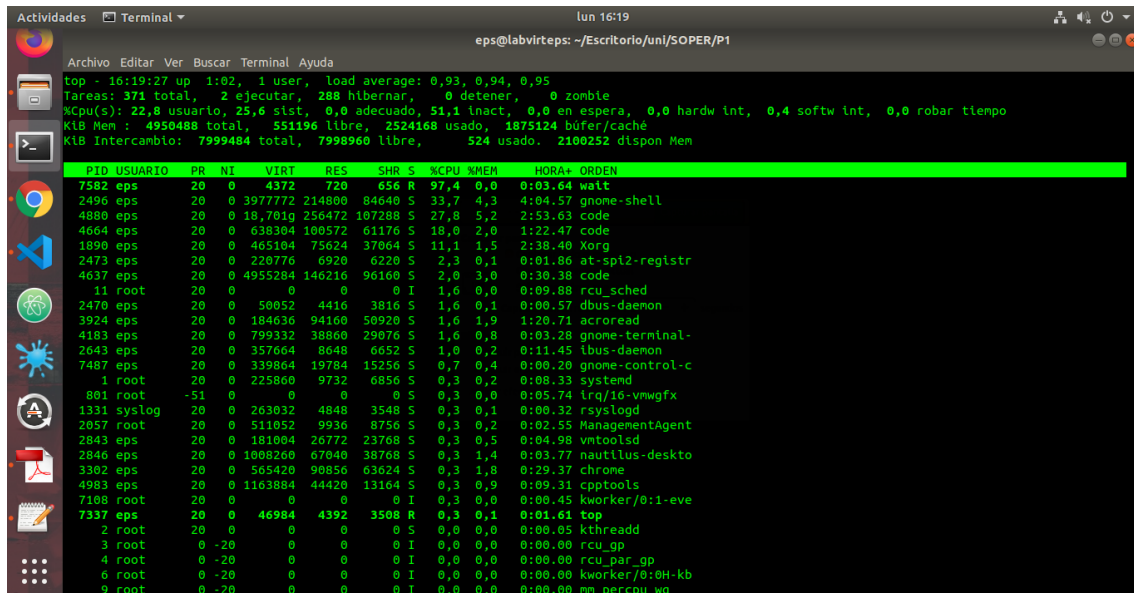
c) Para mostrar las líneas de las listas de la compra de Elena y Pepe usamos el comando “cat ‘lista de la compra Elena.txt’ ‘lista de la compra Pepe’ ”. La salida la usamos de entrada con la pipeline hacia “sort” para ordenar alfabéticamente las líneas obtenidas, mediante otra pipeline eliminamos las repeticiones con el comando “uniq”, contamos el número de elementos con “wc -l” y redirigimos la salida hacia el fichero “num.txt”. El comando es: “cat ‘lista de la compra Elena’ ‘lista de la compra Pepe’ | sort | uniq | wc -l > ‘num.txt’ ”.

Ejercicio 3:

- Al intentar abrir un fichero inexistente recibimos el mensaje “No such file or directory” que corresponde al código de error “2”.
- Si intentamos abrir el fichero “/etc/shadow” recibimos el mensaje “Permission denied”, con código de error “13”.
- Para garantizar que el valor de “errno” es el de “fopen” tenemos que guardar inmediatamente el valor en una variable auxiliar, pues cualquier función que se ejecute después puede modificar el valor de “errno”, aunque la función se ejecute sin errores.

Ejercicio 4:

- Tras ejecutar “top” se puede observar el programa que hemos creado a la cabeza de la lista de procesos con un 0.0% de memoria, un 97.4% de CPU, el valor de la S es R, SHR devuelve el valor “656”, RES “720”, VIRT “4372”, NI “0”, PR “20”, usuario “eps” y, finalmente PID “7582”.



```
top - 16:19:27 up 1:02, 1 user, load average: 0.93, 0.94, 0.95
Tareas: 371 total, 2 ejecutar, 288 hibernar, 0 detener, 0 zombie
%Cpu(s): 22.8 usuario, 25.6 sist, 0.0 adecuado, 51.1 inact, 0.0 en espera, 0.0 hardw int, 0.4 softw int, 0.0 robar tiempo
KiB Mem : 4950488 total, 551196 libre, 2524168 usado, 1875124 búfer/cache
KiB Intercambio: 7999484 total, 7998960 libre, 524 usado, 2100252 dispon Mem

  PID USUARIO PR NI VIRT RES SHR S %CPU %MEM  Hora+ Orden
 7582 eps 20 0 4372 720 656 R 97.4 0.0 0:03.64 wait
2496 eps 20 0 3977772 214800 84640 S 33.7 4.3 4:04.57 gnome-shell
4880 eps 20 0 18,7010 256472 187288 S 27.8 5.2 2:53.63 code
4664 eps 20 0 638384 100572 61176 S 18.0 2.0 1:22.47 code
1890 eps 20 0 465184 75624 37064 S 11.1 1.5 2:38.40 Xorg
2473 eps 20 0 220776 6920 6220 S 2.3 0.1 0:01.86 at-spi2-registr
4637 eps 20 0 4955284 146216 96160 S 2.0 3.0 0:30.38 code
11 root 20 0 0 0 0 I 1.6 0.0 0:09.88 rcu_sched
2470 eps 20 0 50052 4416 3816 S 1.6 0.1 0:00.57 dbus-daemon
3924 eps 20 0 184636 94160 50920 S 1.6 1.9 1:20.71 acroread
4183 eps 20 0 799332 38860 29076 S 1.6 0.8 0:03.28 gnome-terminal-
2643 eps 20 0 357064 8648 6652 S 1.0 0.2 0:11.45 ibus-daemon
7487 eps 20 0 339864 19784 15256 S 0.7 0.4 0:00.20 gnome-control-c
1 root 20 0 225860 9732 6856 S 0.3 0.2 0:08.33 systemd
861 root -SI 0 0 0 0 S 0.3 0.0 0:05.74 irq/16-vmmgfx
1331 syslog 20 0 263032 4848 3548 S 0.3 0.1 0:00.32 rsyslogd
2057 root 20 0 511052 9936 8756 S 0.3 0.2 0:02.55 ManagementAgent
2843 eps 20 0 181004 26772 23768 S 0.3 0.5 0:04.08 vmtotld
2846 eps 20 0 1808260 67040 38768 S 0.3 1.4 0:03.77 nautilus-deskto
3302 eps 20 0 565420 90856 63624 S 0.3 1.8 0:29.37 chrome
4983 eps 20 0 1163884 44420 13164 S 0.3 0.9 0:09.31 cpptools
7108 root 20 0 0 0 0 I 0.3 0.0 0:00.45 kworker/0:1-eve
7337 eps 20 0 46984 4392 3508 R 0.3 0.1 0:01.61 top
2 root 20 0 0 0 0 S 0.0 0.0 0:00.05 kthreadd
3 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 rcu_gp
4 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 rcu_par_gp
6 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 kworker/0:0H-kb
9 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 mm_percpu_wq
```

- Si en vez de usar “clock()” usamos la función “sleep”, entonces no aparece en la lista de procesos cuando usamos “top”

Ejercicio 5:

- Si eliminamos las llamadas a “pthread_join”, el programa no espera a que finalicen los dos hilos tras haberlos creado, por lo que, si el programa finaliza, los hilos también, a pesar de que no hayan terminado su ejecución.
- Si en lugar de llamar a “exit” usamos “pthread_exit”, el programa espera a que terminen los hilos que ha creado y finalmente finaliza.
- Para que sea correcto no esperar a los hijos es necesario desligar los hilos mediante la función “pthread_detach”.

Ejercicio 6:

- a) No se puede saber a priori en que orden se imprimirá el texto, puesto que se desconoce qué proceso terminará antes.
- b) Para que el código, en el proceso hijo, imprima su PID y el de su padre hay que sustituir el "printf()" por "printf("Hijo, id: %d\tPadre, id: %d\n", getpid(), getppid());"
- c) El proceso de arriba corresponde al árbol de la izquierda ya que al hacer "fork()" se obtienen procesos idénticos y concurrentes ejecutando las sentencias siguientes. Para llegar a obtener el árbol de la derecha bastaría con el código siguiente:

```
pid = fork();  
  
if(getppid() != 0 && pid == 0) pid = fork();  
  
if(getppid() != 0 && pid == 0) pid = fork();
```
- d) El código original sí que deja procesos huérfanos, ya que "wait(NULL)" espera solo a uno de los hijos, el primero que termine, el resto se quedarán huérfanos.
- e) Para usar los mínimos cambios posibles tendríamos que meter el "wait(NULL)" dentro del bucle, de esta forma el programa esperará a que el hijo termine para crear otro.

Ejercicio 7:

- a) Cuando se ejecuta este código recibimos por pantalla lo siguiente: "Padre: ", esto sucede porque, aunque el hijo haya modificado la variable, el padre no recibe ese cambio.
- b) Para que el programa no tenga fugas de memoria hay que liberarla en el proceso padre y en el hijo, puesto que al ejecutar "fork()", se crea el proceso hijo con copias de las variables del padre, y pasarán a ser independientes unas de otras, por lo que hay que liberar cada una por su parte.

Ejercicio 8:

- a) Si sustituimos el primer elemento del array "argv" por la cadena "mi-ls" no ocurre nada distinto a la ejecución normal, esto sucede porque "mi-ls" no es un comando existente y, por tanto, no existe ningún fichero con ese nombre, así que no se ejecuta.
- b) Para poder usar la función "execl" tendríamos que cambiar la función "execvp" por "execl("/bin/l", "ls", ".", NULL)", además de cambiar el "perror" a "perror("execl");"

Ejercicio 9:

- a) Ejecutando “cat stat” podemos observar que se trata del ejecutable de “gedit”

Activades

Editor de textos

lun 5:51

eps@labvirtips: /proc/5146

Archivo

Editar

Ver

Buscar

Terminal

Ayuda

```
cpuset    mountinfo    root          timers
cwd       mounts      sched        timerslack_ns
environ   mountstats   schedstat     uid_map
exe       net         sessionid     wchan

eps@labvirtips: /proc/5146$ cat
arch_status fd/          ns/          setgroups
attr/       fdinfo/     numa_maps   smaps
autogroup   gid_map     oom_adj     smaps_rollback
auxv        limits     oom_score    stack
cgroup      loins       oom_score_adj stat
clear_refs  pagemap    statm        status
cmdline     map_files  patch_state  status
comm        maps       personality  syscall
coredump_filter mem         projfd_map   task/
cpuset      mountinfo  root/        timers
cwd/        mounts     sched        timerslack_ns
environ     mountstats schedstat     uid_map
exe         net/       sessionid     wchan

eps@labvirtips: /proc/5146$ cat s
sched      setgroups   stack        status
schedstat  smaps        stat         syscall
sessionid  smaps_rollback statm
eps@labvirtips: /proc/5146$ cat s
sched      setgroups   stack        status
schedstat  smaps        stat         syscall
sessionid  smaps_rollback statm
eps@labvirtips: /proc/5146$ cat stat
stat statm status
eps@labvirtips: /proc/5146$ cat stat
5146 (gedit) S 1628 2062 2062 0 -1 4194304 7235 0 3 0 3647 2279 0 0 2
0 4 0 385492 702193604 13862 18446744073709551615 93958712138656 939
140726719579816 0 0 0 0 4096 0 0 0 17 0 0 45 0 0 93958
13343232 93958712344016 9395871997056 140726719579271 14072671957932
140726719579324 140726719581161 0
eps@labvirtips: /nrcac/5146$ [
```

eps@labvirtips: /proc

Archivo

Editar

Ver

Buscar

Terminal

Ayuda

```
top - 15:51:00 up 4:04, 1 user, load average: 1.00, 0.61, 0.38
Tareas: 395 total, 1 ejecutando, 298 hibernando, 6 detenidas, 0 zom
%CPU(s): 1.2 usuario, 3.2 sist, 0.0 adecuado, 95.2 inactivo, 0.4 en
KIB Mem : 4950488 total, 188588 libre, 2807748 usado, 1954512 bu
KIB Intercambio: 7999484 total, 7998448 libre, 1036 usado, 1734

PID USUARIO PR NI VIRT RES SHR S %CPU %MEM HORA+
2188 eps 20 0 3893184 256864 85572 S 8.3 5.2 8:18.62
1786 eps 20 0 503228 103884 47136 S 6.6 2.1 6:03.84
5146 eps 20 0 685736 5288 32596 S 3.6 1.1 1:02.13
10545 eps 20 0 834428 51680 33640 S 0.7 1.0 0:09.32
1 root 20 0 225632 9256 6612 S 0.3 0.2 0:24.51
765 root -51 0 0 0 0 S 0.3 0.0 0:11.42
1944 root 20 0 96916 12584 10296 S 0.3 0.3 0:19.89
3342 eps 20 0 194268 105716 53448 S 0.3 2.1 3:36.76
3851 eps 20 0 4751924 245952 52476 S 0.3 5.0 0:28.47
```

Abrir

Guardar

comandos.txt

~/Escritorio/WSOPER/P1

```
4) Tras ejecutar top se puede observar el programa a la cabeza de la
lista de procesos con un 0.0 de MEM, un 9.4% de CPU el valor de la S
es R, SHR devuelve el valor de 656, RES 720, VIRT 4372, NI 0, PR
2, usuario EPS y finalmente PID 7583
5) Si en lugar de usar clock(), empleamos la función sleep, entonces
no aparece en la lista de procesos en top
Ejercicio 5:
a) El programa a no espera a que finalicen los dos hilos por lo que
los crea, sigue la ejecución, imprime el final y al retornar la
función principal, ambos hilos del programa finalizan aunque
no hayan terminado de ejecutar la función de esos.
b) Si en lugar de llamar a exit(EXIT_SUCCESS) al finalizar se llama
a pthread_exit(EXIT_SUCCESS), se terminan de ejecutar los hilos y
después finaliza el hilo principal
c) Para que sea correcto no esperar a los hilos es necesario
desligar los hilos mediante la función pthread_detach(h1);
```

- b) Si buscamos en `"/proc/[PID]/cwd"` obtenemos que el directorio actual del proceso es `"/home/eps"`
- c) Se ejecutó con el comando `"gedit/home/eps/Escritorio/uni/SOPER/P1/comandos.txt"`
- d) Con `"cat environ | tr '\0' '\n' | grep 'LANG'"` obtenemos: `LANG=es_ES.UTF-8`
- e) Para ver los procesos hijos, usamos `"cd task; ls"`

Ejercicio 10:

- a) Con “cd /proc/[PID]/fd” podemos observar tres descriptors de fichero (0, 1 y 2), que corresponden a STDIN_FILENO(0), STDOUT_FILENO(1) y STDERR_FILENO(2).
- b) Tras la segunda parada, aparece un nuevo descriptor de fichero con el valor “3”, y tras la tercera parada, otro con el valor “4”.
- c) Tras la cuarta parada se hace un “unlink” del fichero FILE1, lo que desvincula el fichero del programa. En el directorio “fd” podemos observar que el descriptor de fichero sigue estando solo que marcado en rojo y con una etiqueta que indica que esta “borrado”. Sin embargo, esto no será así mientras el programa esté accediendo a él.
- d) Tras la quinta parada, el descriptor de fichero desaparece totalmente, en la sexta parada vuelve a aparecer el descriptor de fichero con valor “3” pero esta vez referenciando al fichero “file3.txt” y, finalmente, con la séptima parada aparece un nuevo descriptor con valor “5”. Con esto podemos deducir que al cerrar un fichero y eliminar su descriptor de fichero, el siguiente descriptor de fichero que se cree obtendrá el valor de este, ocupando los valores intermedios y vacíos que haya de descriptors de fichero.

Ejercicio 11:

a) El mensaje “Yo soy tu padre” aparece dos veces, ya que no se ha limpiado el buffer de salida o se ha usado “\n” antes de usar el “fork()”, una por la secuencia del proceso padre y otra por la del hijo, acompañado del mensaje “Noooooooo”.

b) Como mencionábamos antes, al usar “\n” se limpia el buffer de la salida estándar y, por tanto, ya no aparece en la secuencia del proceso hijo.

c) Al redirigir la salida a un fichero volvemos a encontrarnos con el caso de los dos mensajes de “Yo soy tu padre”, esto es debido a que, como decíamos, “\n” limpia el buffer de salida estándar, pero con la redirección no estamos usando el buffer de salida estándar (line-buffered) sino el “bulky-buffered”, y este no se limpia de esa manera.

d) Para corregir el problema, antes de usar “fork()” deberíamos limpiar el buffer con la función “fflush(stdout)”.

Ejercicio 12:

a) Hemos escrito en el pipe y hemos recibido el string: “Hola a todos!”

b) Si el proceso padre no cierra el extremo de escritura, el programa se queda en un bucle infinito dentro de la función “read”, dado que los descriptores se comparten entre el padre y el hijo. Siempre deberemos cerrar la tubería que no nos preocupa, pues el EOF no se devolverá si no se cierran los extremos explícitamente.

Ejercicio 13:

c) Hemos usado la función “execvp” puesto que no conocemos la cantidad de argumentos que se usarán en cada comando, de esta manera solo nos preocupamos por reservar una cantidad máxima de posibles argumentos, así que no podríamos haber usado otra función.

Cuando ejecutamos la sentencia “sh -c inexistente” obtenemos el string: “Terminated with value <127>”.

Cuando ejecutamos el programa con la sentencia “abort”, recibimos el string: “Terminated with value <134>”.