

Boletín 3: El comando `psplit`

Ampliación de Sistemas Operativos

Dpto. Ingeniería y Tecnología de Computadores (DITEC)

Universidad de Murcia

Curso 2019/2020

1 El comando `psplit`

1. El comando `psplit`

- (4.0p) **EJERCICIO 1:** Implementa el comando interno `psplit`. Dada una secuencia de ficheros `FILEn`, el comando interno `psplit` divide cada fichero `FILEi`, de manera que los ficheros resultantes `FILEi,d` ($d=0,1,2,\dots$) tienen un número máximo de líneas, o bien un número máximo de bytes. Si no se especifica ningún fichero como entrada, `psplit` realiza la misma tarea pero sobre la entrada estándar. En tal caso, los ficheros resultantes empezarán todos por `stdin`.

Ejemplos de uso del comando psplit

● Ejemplo 1: Los ficheros resultantes tienen como máximo una línea:

```
1  simplesh> (echo 0; echo 1) > splitme ; psplit -l 1 splitme
simplesh> wc -l splitme0 splitme1
3  1 splitme0
1 splitme1
5  2 total
simplesh> cat splitme0
7  0
simplesh> cat splitme1
9  1
simplesh> cat splitme0 splitme1 > splitmecopy
11 simplesh> diff splitme splitmecopy
```

● Ejemplo 2: Los ficheros resultantes tienen como máximo un byte:

```
1  simplesh> (echo 0; echo 1) > splitme ; psplit -b 1 splitme
simplesh> wc -c splitme0 splitme1 splitme2 splitme3
3  1 splitme0
1 splitme1
5  1 splitme2
1 splitme3
7  4 total
simplesh> cat splitme0
9  0simplesh> cat splitme1


11 simplesh> cat splitme2
1simplesh> cat splitme3

13
simplesh> cat splitme0 splitme1 splitme2 splitme3 > splitmecopy
15 simplesh> diff splitme splitmecopy
```

- Si se proporciona la opción `-h`, `psplit` debe enviar a `stdout`:

```
1  simplesh> psplit -h
Uso: psplit [-l NLINES] [-b NBYTES] [-s BSIZE] [-p PROCS] [FILE1] [FILE2]...
3      Opciones:
      -l NLINES  Número máximo de líneas por fichero.
5      -b NBYTES  Número máximo de bytes por fichero.
      -s BSIZE   Tamaño en bytes de los bloques leídos de [FILEn] o stdin.
7      -p PROCS   Número máximo de procesos simultáneos.
      -h         Ayuda
```

Procesamiento de opciones

- Dada una función `func(char **argv, int argc)`, `argv` es una lista de `argc` punteros a carácter a las opciones (véase el Tema 1)
- Para procesarlas, se puede usar la función `get_opt`: 

```
void main(int argc, char *argv[]) {
2     int opt, flag, n;
    flag = n = 0;
4     optind = 1;
    while ((opt = getopt(argc, argv, "fn:h")) != -1) {
6         switch (opt) {
            case 'f':
8             flag = 1;
            break;
10            case 'n':
                n = atoi(optarg);
12            break;
            default:
14                fprintf(stderr, "Usage: %s [-f] [-n NUM]\n", argv[0]);
                exit(EXIT_FAILURE);
16        }
    }
18    for(int i = optind; i < argc; i++) printf("%s\n", argv[i]);
}
```

- El tercer parámetro de `get_opt` identifica todas las posibles opciones de manera que `'.'` denota una opción con argumentos

Opciones del comando psplit

- La opción `-l` especifica el número máximo de líneas por fichero:

```
1  simplesh> (echo 1; echo 2; echo 3; echo 4; echo 5) | psplit -l 2
   simplesh> wc -l stdin0 stdin1 stdin2
3    2 stdin0
   2 stdin1
5    1 stdin2
   5 total
```

- La opción `-b` indica el número máximo de bytes por fichero:

```
simplesh> (echo -n 12345) | psplit -b 2
2  simplesh> wc -c stdin0 stdin1 stdin2
   2 stdin0
4   2 stdin1
   1 stdin2
6   5 total
```

- Las opciones `-l` y `-b` son incompatibles:

```
simplesh> (echo 123456) | psplit -l 1 -b 1
2  psplit: Opciones incompatibles
```


Opciones del comando psplit

- Con `-s BSIZE` se especifica el tamaño exacto del *buffer* de lectura/escritura a utilizar con `FILEn` o `stdin`, es decir, SIEMPRE se ha de intentar leer/escribir `BSIZE` bytes en cada operación
- Por defecto, `psplit` divide `FILEn` o `stdin` en ficheros de 1KB:

```
simplesh> dd if=/dev/urandom of=splitme bs=1536 count=1 > /dev/null
2  simplesh> psplit splitme
simplesh> wc -c splitme1 splitme2
4  1024 splitme0
   512 splitme1
6  1536 total
simplesh> cat splitme0 splitme1 > splitmecopy
8  simplesh> diff splitme splitmecopy
```

- El valor de `BSIZE` debe ser un entero tal que $1 \leq BSIZE \leq 2^{20}$

```
simplesh> (echo 123456) | psplit -s 0
2  psplit: Opción -s no válida
```

- La opción `-p PROCS` indica que se pueden procesar hasta `PROCS` ficheros en paralelo mediante la creación de procesos

El comando `psplit` con múltiples FILEn

- Si se especifican múltiples ficheros, `psplit` divide cada uno¹:

```
1  simplesh> dd if=/dev/urandom of=splitme1 bs=1024 count=1 > /dev/null
2  simplesh> cp splitme1 splitme2
3  simplesh> psplit -b 500 splitme1 splitme2
4  simplesh> wc -c splitme10 splitme11 splitme12
      500 splitme10
      500 splitme11
       24 splitme12
      1024 total
5  simplesh> wc -c splitme20 splitme21 splitme22
      500 splitme20
      500 splitme21
       24 splitme22
      1024 total
6  simplesh> cat splitme10 splitme11 splitme12 > splitmecopy
7  simplesh> diff -s splitme splitmecopy
8  Los ficheros splitme y splitmecopy son idénticos
```

¹ Nótese que en este ejemplo el contenido de los ficheros FILEn es binario, es decir, no es texto. En consecuencia, las funciones de manipulación de cadenas como `strcat()` o `strcpy()` **NO** pueden utilizarse para procesar la entrada en `psplit`.

El comando psplit sin FILEn

- Si no se especifica FILEn, psplit divide stdin:

```
1  simplesh> (echo 0; echo 1) | tee splitme | psplit -l 1
2  simplesh> wc -c stdin0 stdin1
3  2 stdin0
4  2 stdin1
5  4 total
6  simplesh> cat stdin0
7  0
8  simplesh> cat stdin1
9  1
10 simplesh> cat stdin0 stdin1 > splitmecopy
11 simplesh> diff splitme splitmecopy
12
13 simplesh> (echo 0; echo 1) | tee splitme | psplit -b 1
14 simplesh> wc -c stdin0 stdin1 stdin2 stdin3
15 1 stdin0
16 1 stdin1
17 1 stdin2
18 1 stdin3
19 4 total
20 simplesh> cat stdin0
21 0simplesh> cat stdin1
22
23 simplesh> cat stdin2
24 1simplesh> cat stdin3
25
26 simplesh> cat stdin0 stdin1 stdin2 stdin3 > splitmecopy
27 simplesh> diff splitme splitmecopy
```

Implementación del comando `psplit` (I)

- Para implementar `psplit`:
 - Únicamente se pueden usar llamadas al sistema POSIX o funciones de la biblioteca estándar de C (C11)
 - Se deben considerar, al menos, las llamadas POSIX y/o glibc:
 - `getopt()`
 - `open()` / `close()`
 - `read()` / `write()`
 - `fsync()` / `close()`
 - `fork()` / `exit()`
 - Se debe escribir la función `run_psplit()` e insertar llamadas a la misma en `simplesh.c` donde sea necesario

Implementación del comando `psplit` (II)

- 1 Procesar las opciones con `getopt()` (man 3 `getopt`)
 - **Nota:** Antes de volver a usar `getopt()`, se debe inicializar `optind` a 1
- 2 Comprobar la validez de las opciones
- 3 Procesar archivos `FILEn`, o `stdin` si no hay `FILEn`, secuencialmente
 - Leer `BSIZE` bytes y escribirlos en el fichero de salida, hasta que se haya alcanzado el número máximo de líneas o bytes, o el final del fichero de entrada (`FILEn` o `stdin`)
 - Cada vez que se alcanza el número máximo de líneas o bytes, se debe crear un nuevo fichero de salida
- 4 Cada vez que se cierra un fichero, se debe *sincronizar* con `fsync()`
- 5 Utilizar `read()` y `write()` para realizar las operaciones de lectura y escritura, es decir, en ningún caso se deben usar funciones de la biblioteca estándar de C como `fread()` o `fwrite()`
- 6 Asegurar que `psplit` trata correctamente las escrituras parciales

Implementación del comando `psplit` (III)

- ❶ Procesar los archivos `FILEn`, o `stdin` si no hay `FILEn`, en paralelo
 - Siendo `BPROCS` el número máximo de procesos *en vuelo*
 - Si no se ha alcanzado `PROCS`, crear un nuevo proceso
 - Leer `BSIZE` bytes y escribirlos en el fichero de salida, hasta que se haya alcanzado el número máximo de líneas o bytes, o el final del fichero de entrada (`FILEn` o `stdin`)
 - Cada vez que se alcanza el número máximo de líneas o bytes, se debe crear un nuevo fichero de salida
 - Si se ha alcanzado `PROCS`, esperar al proceso más antiguo
 - Esperar a que terminen todos los procesos en orden de antigüedad