PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

2da práctica (tipo a)
(Primer semestre de 2015)

Horario 0781: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: No se puede usar ningún material de consulta.
La presentación, la ortografía y la gramática influirán en la calificación.
Puntaje total: 20 puntos

Pregunta 1 (5 puntos – 25 min.) En sistemas Unix hay muchos editores de texto y cada uno los elige según su gusto. El editor clásico **ed** fue escrito originalmente por Ken Thompson en 1971 (hace 44 años) y este editor de texto está presente obligatoriamente en todos sistemas Unix. Aquí el está en el último Linux Mint 17.1 Rebecca:

```
vk@lanfier ~/clases/so/progs $ which ed
/bin/ed
vk@lanfier ~/clases/so/progs $ ls -l /bin/ed
-rwxr-xr-x 1 root root 47712 jul 16  2013 /bin/ed
```

y en el último OS X Yosemite 10.10.3:

```
zurbagan:so vk$ which ed
/bin/ed
zurbagan:so vk$ ls -l /bin/ed
-rwxr-xr-x  1 root  wheel  49904 Sep  9  2014 /bin/ed
```

La versión avanzada de este editor estándar fue **ex** (de ***Ex****tended*), escrito originalmente por Bill Joy en 1976:

```
$ which ex
/usr/bin/ex
$ ls -li /usr/bin/ex
24117564 lrwxrwxrwx 1 root root 20 ago  8  2014 /usr/bin/ex -> /etc/alternatives/ex
$ ls -liL /usr/bin/ex
24118700 -rwxr-xr-x 1 root root 884360 ene  2  2014 /usr/bin/ex
$ ls -li /etc/alternatives/ex
12190082 lrwxrwxrwx 1 root root 17 ago  8  2014 /etc/alternatives/ex -> /usr/bin/vim.tiny
$ ls -liL /etc/alternatives/ex
24118700 -rwxr-xr-x 1 root root 884360 ene  2  2014 /etc/alternatives/ex
$ ls -li /usr/bin/vim.tiny
24118700 -rwxr-xr-x 1 root root 884360 ene  2  2014 /usr/bin/vim.tiny
```

Ese mismo año Bill Joy escribió la versión **vi**sual del editor – **vi**, puesto como *hard link* a **ex** en 2BSD (*Second Berkeley Software Distribution*, mayo de 1979). Con los años este editor se convirtió en el editor estándar de Unix *de facto*.

```
$ which vi
/usr/bin/vi
$ ls -li /usr/bin/vi
24118697 lrwxrwxrwx 1 root root 20 ago  8  2014 /usr/bin/vi -> /etc/alternatives/vi
$ ls -liL /usr/bin/vi
24118700 -rwxr-xr-x 1 root root 884360 ene  2  2014 /usr/bin/vi
$ ls -li /etc/alternatives/vi
12190209 lrwxrwxrwx 1 root root 17 ago  8  2014 /etc/alternatives/vi -> /usr/bin/vim.tiny
$ ls -liL /etc/alternatives/vi
24118700 -rwxr-xr-x 1 root root 884360 ene  2  2014 /etc/alternatives/vi
```

En 1988 Bram Moolenaar escribió un clon de **vi** – **vim** (primero de ***vi IM****itation*, después de ***vi IM****proved*). En 2006 **vim** fue votado como el editor más popular entre los lectores de la revista *Linux Journal*. Mientras que **vi** fue originalmente disponible solamente en los sistemas operativos Unix, **vim** puede ejecutarse en muchos sistemas: AmigaOS, Atari MiNT, BeOS, DOS, Windows a partir de Windows 95, OS/2, MorhOS, OpenVMS, QNX, RISC OS, GNU/Linux, BSD y Classic MAC OS. También Vim viene con cada copia de Apple OS X. **vim** está disponible para Android y iOS.

```
$ which vim
/usr/bin/vim
$ ls -li /usr/bin/vim
24126100 lrwxrwxrwx 1 root root 21 abr 23 23:35 /usr/bin/vim -> /etc/alternatives/vim
$ ls -liL /usr/bin/vim
24126056 -rwxr-xr-x 1 root root 2191736 ene  2  2014 /usr/bin/vim
$ ls -li /etc/alternatives/vim
12190167 lrwxrwxrwx 1 root root 18 abr 23 23:35 /etc/alternatives/vim -> /usr/bin/vim.basic
$ ls -li /usr/bin/vim.basic
24126056 -rwxr-xr-x 1 root root 2191736 ene  2  2014 /usr/bin/vim.basic
```

En las ordenes proporcionadas aquí se mencionan varios programas de edición de texto instaladas en Linux Mint particular. La opción **`-L`** de la orden **`ls`** realiza "*dereference*" del enlace. El tercer campo de salida de la orden **`ls`** (el separador de campos es el espacio) indica la cantidad de enlaces al archivo.

a) (3 puntos – 15 min.) Indique las rutas (*path*) de los archivos de los programas editores de texto instalados, con todos sus nombres para invocarlos.

b) (1 punto – 5 min.) Si algunos archivos de programas tienen varios nombres, ¿por qué el programa **`ls`** indica siempre un sólo enlace?

c) (1 punto – 5 min.) En el directorio **`/usr/bin/`** hay algunos archivos ejecutables pero de tamaño demasiado pequeño para un archivo ejecutable:

```
$ ls -l /usr/bin/vim*
-rwxr-xr-x 1 root root 2191736 ene  2  2014 /usr/bin/vim.basic
-rwxr-xr-x 1 root root    1051 dic  4 22:29 /usr/bin/vimdot
-rwxr-xr-x 1 root root  884360 ene  2  2014 /usr/bin/vim.tiny
-rwxr-xr-x 1 root root    2084 ene  2  2014 /usr/bin/vimtutor
```

¿Pueden estos archivos ejecutables ser simplemente los archivos de texto ASCII? Explíquelo.

Pregunta 2 (5 puntos – 25 min.) "The advantage of using `popen` and `pclose` is that the interface is much simpler and easier to use."

```
popen(3) - Linux man page

Name
       popen, pclose - pipe stream to or from a process

Synopsis
       #include <stdio.h>

       FILE *popen(const char *command, const char *type);

The popen() function opens a process by creating a pipe, forking, and invoking the shell. Since a pipe is by
definition unidirectional, the type argument may specify only reading or writing, not both; the resulting
stream is correspondingly read-only or write-only.

The command argument is a pointer to a null-terminated string containing a shell command line. This command is
passed to /bin/sh using the -c flag; interpretation, if any, is performed by the shell. The type argument is a
pointer to a null-terminated string which must contain either the letter 'r' for reading or the letter 'w' for
writing.

The return value from popen() is a normal standard I/O stream in all respects save that it must be closed with
pclose() rather than fclose(3). Writing to such a stream writes to the standard input of the command; the
command's standard output is the same as that of the process that called popen(), unless this is altered by
the command itself. Conversely, reading from a "popened" stream reads the command's standard output, and the
command's standard input is the same as that of the process that called popen().
```

Here is an example showing how to use `popen` and `pclose` to filter output through another program, in this case the paging program `more`.

```
#include <stdio.h>
#include <stdlib.h>

void
write_data(FILE *stream)
{
  int i;
  for (i = 0; i < 100; i++) fprintf (stream, "%d\n", i);
  if (ferror(stream)) {
      fprintf (stderr, "Output to stream failed.\n");
      exit (EXIT_FAILURE);
  }
}
```

```
int
main (void)
{
  FILE *output;

  output = popen(...);
  if (!output) {
      fprintf (stderr, "incorrect parameters or too many files.\n");
      return EXIT_FAILURE;
  }
  write_data(output);
  if (pclose(output) != 0) {
      fprintf (stderr, "Could not run more or other error.\n");
  }
  return EXIT_SUCCESS;
}
```

a) (3 puntos – 15 min.) Complete la sentencia `output = popen(...);`.

b) (2 puntos – 10 min.) Indique la secuencia de llamadas al sistema (en el orden correcto) que realiza la función `popen()`.

Pregunta 3 (5 punto – 25 min., *Ben-Ari, PCDP/2E*) Para valores positivos de $K > 10$, ¿cuáles son todos valores finales posibles de *n* en el siguiente algoritmo? Describe los 4 casos de secuencias de ejecución que producen los valores extremos, del medio y de un tercio del rango.

```
int n = 0;

Hilo P                  Hilo Q

int t1;                 int t2;

P1: do K times          Q1: do K times
P2: t1 = n              Q2: t2 = n
P3: n = t1 + 1          Q3: n = t2 - 1
```

Pregunta 4 (5 puntos – 25 min.) El algoritmo de Peterson:

```
flag[0]   = 0;
flag[1]   = 0;
turn;

P0: flag[0] = 1;
    turn = 1;
    while (flag[1] == 1 && turn == 1)
    {
          // busy wait
    }
    // critical section
       ...
    // end of critical section
    flag[0] = 0;

P1: flag[1] = 1;
    turn = 0;
    while (flag[0] == 1 && turn == 0)
    {
          // busy wait
    }
    // critical section
       ...
    // end of critical section
    flag[1] = 0;
```

El proceso *P0* tiene su sección no-crítica muy corta: al terminar la sección crítica, el de inmediato quita y otra vez coloca su bandera en `flag[0]` intentando otra vez entrar en su sección crítica. La pregunta es si algoritmo de Peterson garantiza o no la ausencia de inanición (*starvation*) de otro proceso. Explíquelo.

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 17.1.

Profesor del curso: (0781) V. Khlebnikov

Pando, 24 de abril de 2015