PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
2da práctica (tipo a)
(Segundo 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 (3 puntos – 15 min.) ¿Cómo funciona el siguiente programa y cuál es su propósito? Si no hay lecturas de la tubería, ¿por qué no se cierra el descriptor correspondiente?

```
$ cat -n khavbsal.c
     1  #include <stdio.h>
     2  #include <stdlib.h>
     3  #include <unistd.h>
     4  #include <signal.h>
     5
     6  int count = 0;
     7  void alrm_action(int);
     8
     9  int
    10  main(void)
    11  {
    12      int p[2];
    13      char c = 'x';
    14
    15      if (pipe(p) == -1) {perror("pipe"); exit(EXIT_FAILURE); }
    16      signal(SIGALRM, alrm_action);
    17      while (1) {
    18          alarm(1);   /* set 1 sec alarm */
    19          write(p[1],&c,1);
    20          alarm(0);   /* cancel alarm */
    21          if ((++count % 1024) == 0)
    22              printf("%d characters in pipe\n", count);
    23      }
    24  }
    25
    26  void
    27  alrm_action(int sig)
    28  {
    29      printf(...);
    30      exit(EXIT_SUCCESS);
    31  }
```

Pregunta 2 (7 puntos – 35 min.) (*TLPI by M. Kerrisk*) Analice el siguiente programa:

```
$ cat -n pipe_sync.c
     1  #include <time.h>
     2  #include <stdarg.h>
     3  #include <stdio.h>
     4  #include <stdlib.h>
     5  #include <unistd.h>
     6
     7  void usageErr(const char *format, ...);
     8  char *currTime(const char *format);
     9
    10  int
    11  main(int argc, char *argv[])
    12  {
    13      int pfd[2];
    14      int j, dummy;
    15
    16      if (argc < 2 || strcmp(argv[1], "--help") == 0)
    17          usageErr("%s sleep-time...\n", argv[0]);
    18
    19      setbuf(stdout, NULL);   /* Make stdout unbuffered, since we
    20                                 terminate child with _exit() */
```

```
21      printf("%s  Parent started\n", currTime("%T"));
22      if (pipe(pfd) == -1) perror("pipe");
23      for (j = 1; j < argc; j++) {
24          switch (fork()) {
25          case -1:
26              perror("fork");
27          case 0:
28              if (close(pfd[0]) == -1) perror("close");
29              /* Child does some work */
30              sleep(atoi(argv[j]));
31              printf("%s  Child %d (PID=%ld) closing pipe\n",
32                      currTime("%T"), j, (long) getpid());
33              if (close(pfd[1]) == -1) perror("close");
34              /* Child now carries on to do other things... */
35              _exit(EXIT_SUCCESS);
36          default:
37              break;
38          }
39      }
40      if (close(pfd[1]) == -1) perror("close");
41      /* Parent may do other work... */
42      if (read(pfd[0], &dummy, 1) != 0) perror("parent didn't get EOF");
43      printf("%s  Parent ready to go\n", currTime("%T"));
44      /* Parent can now carry on to do other things... */
45      exit(EXIT_SUCCESS);
46  }
47
48  void
49  usageErr(const char *format, ...)
50  {
51      va_list argList;
52
53      fflush(stdout);             /* Flush any pending stdout */
54
55      fprintf(stderr, "Usage: ");
56      va_start(argList, format);
57      vfprintf(stderr, format, argList);
58      va_end(argList);
59
60      fflush(stderr);             /* In case stderr is not line-buffered */
61      exit(EXIT_FAILURE);
62  }
63
64  char *
65  currTime(const char *format)
66  {
67      static char buf[1000];      /* Nonreentrant */
68      time_t t;
69      size_t s;
70      struct tm *tm;
71
72      t = time(NULL);
73      tm = localtime(&t);
74      if (tm == NULL)
75          return NULL;
76      s = strftime(buf, 1000, (format != NULL) ? format : "%c", tm);
77      return (s == 0) ? NULL : buf;
78  }
```

a) (4 puntos – 20 min.) Complete lo indicado con "**..**" en la salida obtenida de la siguiente ejecución del programa:

```
$ ./pipe_sync 4 2 6
19:59:29  Parent started
19:59:..  Child .. (PID=..) closing pipe
19:59:..  Child .. (PID=..) closing pipe
19:59:..  Child .. (PID=..) closing pipe
19:59:..  Parent ready to go
```

b) (3 puntos – 15 min.) Si ningún hijo no escribe nada en la tubería, entonces para que el padre hace la llamada al sistema `read()`? ¿Qué sucede si el padre no cierra su lado de escritura a la tubería? ¿Qué sucede si los hijos no cierran su lado de lectura de la tubería?

Pregunta 3 (5 puntos – 25 min.) Considere el siguiente código:

```
$ cat -n threads_a.c
     1  #include <pthread.h>
     2  #include <stdlib.h>
     3  #include <stdio.h>
     4
     5  int myglobal;
     6  pthread_t mythread1, mythread2;
     7
     8  void *
     9  thread_function1(void *arg)
    10  {
    11      int i,j;
    12      for (i=0; i<20; i++) {
    13          j=myglobal;
    14          j++;
    15          fprintf(stderr,".");
    16          sleep(1);
    17          myglobal=j;
    18      }
    19      pthread_exit(0);
    20  }
    21
    22  void *
    23  thread_function2(void *arg)
    24  {
    25      int i,j;
    26      for (i=0; i<20; i++) {
    27          j=myglobal;
    28          j++;
    29          fprintf(stderr,":");
    30          myglobal=j;
    31      }
    32      pthread_exit(0);
    33  }
    34
    35  int
    36  main(void)
    37  {
    38      int i;
    39      if (pthread_create(&mythread1,NULL,thread_function1,NULL)) {
    40          printf("error creating thread 1"); abort(); }
    41      if (pthread_create(&mythread2,NULL,thread_function2,NULL)) {
    42          printf("error creating thread 2"); abort(); }
    43      for (i=0; i<20; i++) {
    44          myglobal += 1;
    45          fprintf(stderr,"o");
    46      }
    47      printf("\nmyglobal equals %d\n",myglobal);
    48      exit(0);
    49  }
```

En la ejecución se obtiene la siguiente salida:

```
$ gcc threads_a.c -o threads_a -lpthread
$ time ./threads_a    # en este caso time presenta el tiempo usado por el proceso
ooooooo.:::::::::::::ooooooooooooo
:myglobal equals 33
::::::
real    0m0.001s    # 1 ms
user    0m0.000s
sys     0m0.001s
```

a) (1 punto – 5 min.) ¿Cuál fue el número de la línea de código que se ejecutó la última?

b) (1 punto – 5 min.) ¿Qué modificación fue hecha en el programa para obtener la siguiente salida?

```
$ gcc threads_b.c -o threads_b -lpthread
$ time ./threads_b
.:::::::::::::::::::o.o.o.o.o..oo.o.o.o..oo.o..o.o.oo.o.o.o
myglobal equals 20

real    0m20.003s
user    0m0.000s
sys     0m0.001s
```

c) (1 punto – 5 min.) Después de una modificación (de una línea) el programa fue ejecutado con el depurador **`gdb`**. ¿Cuál fue la modificación?

```
$ gcc threads_c.c -o threads_c -lpthread
$ gdb ./threads_c
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
...
(gdb) run
Starting program: /home/vk/clases/so/progs/threads_c
[Depuración de hilo usando libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Nuevo Thread 0x7ffff77f6700 (LWP 18830)]
.[Nuevo Thread 0x7ffff6ff5700 (LWP 18831)]
::::::::::::::::::::[Thread 0x7ffff6ff5700 (LWP 18831) terminado]
[Thread 0x7ffff77f6700 (LWP 18830) terminado]
[Inferior 1 (process 18826) exited normally]
(gdb) quit
```

d) (2 puntos – 10 min.) En la siguiente versión del programa indique cómo sincronizar la ejecución de los hilos (sin usar mutex) para obtener el siguiente resultado con el tiempo de ejecución de unos milisegundos:

```
$ gcc threads_d.c -o threads_d -lpthread
$ time ./threads_d
....................::::::::::::::::::::oooooooooooooooooooo
myglobal equals 60

real    0m0.001s
user    0m0.000s
sys     0m0.001s
```

<u>Pregunta 4</u> (5 puntos – 25 min.) Para `K` > 10, ¿cuáles son todos posibles valores finales de `n` en el siguiente algoritmo? ¿Cómo se obtienen los valores extremos, el valor del medio y un valor `x` cualquiera?

```
integer n := 0
Hilo P                      Hilo Q
     integer t1                  integer t2
P1:  do K times             Q1:  do K times
P2:      t1 := n            Q2:      t2 := n
P3:      n := t1 + 1        Q3:      n := t2 - 1
```

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 17.2 Rafaela.

Profesor del curso: (0781) V. Khlebnikov

Pando, 25 de septiembre de 2015