PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

1ra práctica (tipo a)
(Segundo semestre de 2019)

Horario 0781: prof. V. Khlebnikov
Horario 0782: prof. F. Solari A.

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.
La práctica debe ser desarrollada en el cuadernillo usando lapicero.
Lo escrito con lápiz NO será evaluado.
Puntaje total: 20 puntos

Pregunta 1 (4 puntos – 20 min.)
a) (1 punto – 5 min.) ¿Cómo un proceso de usuario obtiene un servicio del núcleo? ¿Cómo lo hace un dispositivo?
b) (1 punto – 5 min.) Si la velocidad de transferencia al disco externo es de 50 MB/s, ¿cuánto tiempo tomará hacer un respaldo de un disco de 2 TB? (Sin calculadora, usando las potencias de 10 porque los fabricantes de discos indican sus capacidades en base 10.)
c) (1 punto – 5 min.) Según la regla de Duff, un nanosiglo es cerca de π segundos: 1 año = 365.25 días, 1 siglo = 3,155,760,000 segundos, 1 nanosiglo = 3.15576 segundos. Si se necesitan 10^{15} operaciones y se ejecutan 10^{6} operaciones en 1 segundo, ¿cuántos años tomará el cálculo? De otro lado, la regla útil para recordar es que 1 año son unos … (en segundos con el prefijo más adecuado).
d) (0,5 puntos – 3 min.) Si dos procesos ejecutan el mismo programa, ¿qué segmentos de memoria de un proceso pueden ser reusados en otro proceso y cuáles no? ¿Por qué?
e) (0,5 puntos – 3 min.) ¿Qué permisos tendrá el archivo después de ejecutar la siguiente orden: `$ chmod 0750 foo`?

Pregunta 2 (4 puntos – 20 min.) Responda a las siguientes cuestiones:
a) (2 puntos – 10 min.) La estructura **monolítica** es la más común en muchos sistemas operativos, mientras que la estructura de ***microkernel*** tiene características, digamos, deseables. Describa muy concretamente la estructura de **microkernel** (1 punto) y señale alguna ventaja y una desventaja (1 punto).
b) (2 puntos – 10 min.) Diferencie entre un ***type 1 hypervisor*** y un ***type 2 hypervisor*** en cuánto a su acceso al hardware, y a los recursos de un sistema operativo.

Pregunta 3 (6 puntos – 30 min.) Considere los siguientes programas (de Majd F. Sakr, presentados en las diapositivas de las clases, modificados) y los resultados de sus ejecuciones:

```
$ cat -n 2019-2_pr1.c
     1  #include <sys/types.h>
     2  #include <sys/wait.h>
     3  #include <stdlib.h>
     4  #include <stdio.h>
     5  #include <unistd.h>
     6
     7  #define N 10
     8
     9  int
    10  main(void) {
    11      pid_t pid[N];
    12      int i, status;
    13
    14      for (i = 0; i < N; i++)
    15          if ((pid[i] = fork()) == 0) {
    16              pid[i] = getpid();
    17              pid[i] %= 3;
    18              pid[i] = getpid()%pid[i];
    19              exit(100+i);
    20          }
    21      sleep(N);
    22      for (i = 0; i < N; i++) {
    23          pid_t wpid = waitpid(pid[i], &status, 0);
    24          if (WIFEXITED(status))
    25              printf("Child %d terminated with exit status %d\n",
    26                     wpid, WEXITSTATUS(status));
    27          else
    28              printf("Child %d terminate abnormally\n", wpid);
    29      }
```

```
    30  }
$ gcc 2019-2_pr1.c -o 2019-2_pr1
$ ./2019-2_pr1
Child 7505 terminated with exit status 100
Child 7506 terminate abnormally
Child 7507 terminated with exit status 102
Child 7508 terminated with exit status 103
Child 7510 terminated with exit status 104
Child 7511 terminated with exit status 105
Child 7512 terminate abnormally
Child 7513 terminated with exit status 107
Child 7514 terminated with exit status 108
Child 7515 terminate abnormally
```

a) (2 puntos – 10 min.) ¿Por qué la macro `WIFEXITED`, que analiza la variable `status` (línea 24), devuelve *false* cuando el proceso termina anormalmente? ¿Por qué algunos procesos terminaron anormalmente?
b) (1 punto – 5 min.) Algunos procesos terminaron normalmente y otros no. Pero, ¿qué puede decir usted sobre los procesos *zombie* respecto a la ejecución de este programa?

```
$ cat -n 2019-2_pr1a.c
     1  #include <sys/types.h>
     2  #include <sys/wait.h>
     3  #include <stdlib.h>
     4  #include <stdio.h>
     5  #include <unistd.h>
     6
     7  #define N 10
     8
     9  int ccount = N;
    10
    11  void
    12  child_handler(int sig) {
    13      int status;
    14      pid_t pid;
    15
    16      while ((pid = wait(&status)) > 0) {
    17          ccount--;
    18          printf("Received signal %d from process %d\n", sig, pid);
    19          fflush(stdout);
    20      }
    21  }
    22
    23  int
    24  main(void) {
    25      pid_t pid[N];
    26      int i,j;
    27
    28      signal(SIGCHLD, child_handler);  /* This signal is sent (by the kernel)
    29                                          to a parent process when one of its
    30                                          children terminates (either by calling
    31                                          exit() or as a result of being killed
    32                                          by a signal). It may also be sent to
    33                                          a process when one of its children
    34                                          is stopped or resumed by a signal. */
    35      for (i = 0; i < N; i++)
    36          if ((pid[i] = fork()) == 0) {
    37              sleep(1);
    38              for (j = 0; j < ccount; j++)
    39                  execl("/bin/date","date","-d","11:00 tomorrow",NULL);
    40              exit(17+i);
    41          }
    42      while (ccount > 0)
    43          pause(); /* causes the calling process to sleep until a signal
    44                      is delivered that either terminates the process or
    45                      causes the invocation of a signal-catching function. */
    46  }
$ gcc 2019-2_pr1a.c -o 2019-2_pr1a
$ ./2019-2_pr1a
vie set 13 11:00:00 -05 2019
vie set 13 11:00:00 -05 2019
vie set 13 11:00:00 -05 2019
vie set 13 11:00:00 -05 2019
vie set 13 11:00:00 -05 2019
vie set 13 11:00:00 -05 2019
vie set 13 11:00:00 -05 2019
Received signal 17 from process 20600
...
```

c) (1 punto – 5 min.) ¿Cuántas líneas tendrá la salida producida por este programa?
d) (1 punto – 5 min.) ¿Para qué se usa `fflush()` (la línea 19)?
e) (1 punto – 5 min.) ¿Cuáles serán los códigos de retorno con los cuales terminarán los procesos hijos?

<u>Pregunta 4</u> (6 puntos – 30 min.) El siguiente código en lenguage C, es la parte principal de una implementación de la *función de librería C* ***popen()*:**

```
$ cat -n P1-p4-2019-2.c
 1  static int pids[OPEN_MAX];
 2
 3  FILE *
 4  popen(command, type)
 5  CONST char *command;
 6  CONST char *type;
 7  {
 8          int piped[2];
 9          int Xtype = *type == 'r' ? 0 : *type == 'w' ? 1 : 2;
10          int pid;
11
12          if (Xtype == 2 ||
13              pipe(piped) < 0 ||
14              (pid = fork()) < 0) return 0;
15
16          if (pid == 0) {
17                  /* child */
18                  register int *p;
19
20                  for (p = pids; p < &pids[OPEN_MAX]; p++) {
21                          if (*p) close((int)(p - pids));
22                  }
23                  close(piped[Xtype]);
24                  dup2(piped[!Xtype], !Xtype);
25                  close(piped[!Xtype]);
26                  execl("/bin/sh", "sh", "-c", command, (char *) 0);
27                  exit(127);      /* like system() */
28          }
29
30          pids[piped[Xtype]] = pid;
31          close(piped[!Xtype]);
32          return fdopen(piped[Xtype], type);
33  }
34
35  int
36  pclose(stream)
37  FILE *stream;
38  {
39          int fd = fileno(stream);
40          wait_arg status;
41          int wret;
42
43          void (*intsave)(int) = signal(SIGINT, SIG_IGN);
44          void (*quitsave)(int) = signal(SIGQUIT, SIG_IGN);
45
46          fclose(stream);
47          while ((wret = wait(&status)) != -1) {
48                  if (wret == pids[fd]) break;
49          }
50          if (wret == -1) ret_val = -1;
51          signal(SIGINT, intsave);
52          signal(SIGQUIT, quitsave);
53          pids[fd] = 0;
54          return ret_val;
55  }
```

a) (1 punto – 5 min.) Haga un esquema de los procesos involucrados, la relación con alguna tubería y el sentido de lectura o escritura, cuando en un programa se usa la función *popen()* como, por ejemplo, con *filest1=popen("ls -l",'r');*
b) (1 punto – 5 min.) ¿Qué ocurre entre las líneas 12 a 14? Explique las posibilidades.
c) (3 puntos – 15 min.) Explique el manejo de descriptores de archivo entre las líneas 20 a 25, considerando el mismo ejemplo de invocación del comando "ls -l" con opción 'r'.
d) (1 punto – 5 min.) Indique el funcionamiento general de *pclose(filest1)*.

La práctica ha sido preparada por FS (2,4) y VK (1,3)
con LibreOffice Writer en Linux Mint 19.2 "Tina"

Profesores del curso: (0781) V. Khlebnikov
(0782) F. Solari A.

Pando, 13 de septiembre de 2019