PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
Examen 1
(Segundo semestre de 2019)

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

Duración: 3 horas
Nota: No se puede usar ningún material de consulta.
La presentación, la ortografía y la gramática influirán en la calificación.
El examen debe ser desarrollado en el cuadernillo usando lapicero.
Lo escrito con lápiz será considerado como borrador y NO será evaluado.
Puntaje total: 20 puntos

Pregunta 1 (4 puntos – 32 min.)
a) (1 punto – 8 min.) ¿Cuáles son las definiciones más comunes o aceptadas de un Sistema Operativo?

b) (2 puntos – 16 min.) Mencione dos llamadas al sistema para cada categoría siguiente: Manejo de Procesos, Manejo de Archivos, Manejo de File System, Miscelánea (algunas otras llamadas al sistema) (0,5 puntos cada categoría)

c) (1 punto – 8 min.) Dada la estructura de Microkernel de un sistema operativo, señale dos ventajas y dos desventajas desde el punto de vista de las llamadas al sistema hechas por un proceso.

Pregunta 2 (3 puntos – 24 min.) (*Keith Haviland – Ben Salama*) Analice el siguiente código en Lenguaje C:

```
fsolari@piscis:~/testjoin$ cat -n testjoin.c
     1  #include <stdio.h>
     2  #include <stdlib.h>
     3  #include <sys/types.h>
     4  #include <sys/wait.h>
     5  #include <unistd.h>
     6
     7  fatal(s) /* print error message and die */
     8  char *s;
     9  {
    10     perror(s);
    11     exit(1);
    12  }
    13
    14  int join(com1, com2)
    15  char *com1[],*com2[];
    16  {
    17     int p[2], status;
    18
    19          /* create .....................*/
    20          switch(fork()) {
    21                  case -1:
    22                          fatal("1st fork call in join");
    23                  case  0:
    24                          break;
    25                  default:
    26                          wait(&status);
    27                          return(status);
    28          }
    29          /* remainder ..................*/
    30
    31          /* make .......................*/
    32          if(pipe(p) < 0)
    33                  fatal("pipe call in join");
    34
    35          /*create .....................*/
    36          switch(fork()) {
    37                  case -1:                /*.............*/
    38                          fatal("2nd fork call in join");
    39                  case 0:
    40                          close(1);       /*.............*/
    41                          dup(p[1]);      /*.............*/
```

```
42
43                          close(p[1]);/*.............*/
44                          close(p[0]);
45
46                          execvp(com1[0],com1);
47                          /*.............*/
48                          fatal("1st execvp call in join");
49                  default:
50                          close(0);  /*.............*/
51                          dup(p[0]); /*.............*/
52
53                          close(p[0]); /*.............*/
54                          close(p[1]);
55
56                          execvp(com2[0],com2);
57                          /*.............*/
58                          fatal("1st execvp call in join");
59          }
60  }
61
62  int main()
63  {
64
65     char *one[4], *two[3];
66     int ret;
67          /*.............*/
68     one[0]="ls";
69     one[1]="-l";
70     one[2]="/usr/lib";
71     one[3]=NULL;
72     /*.............*/
73     two[0]="grep";
74     two[1]="java";
75     two[2]=(char *)0;
76          /*.............*/
77     ret=join(one,two);
78     printf("join returned %d\n",ret);
79     exit(0);
80  }
```

a) (0,5 puntos – 4 min.) Agregue los comentarios de la función *main*. (líneas 62 a 80).

b) (1 punto – 8 min.) Agregue los comentarios de la función *join*. (líneas 14 a 60).

c) (1 punto – 8 min.) Explique cuantos procesos se tienen durante la ejecución de *testjoin,* el funcionamiento de cada proceso y como se relacionan (haga un esquema si lo cree necesario).

d) (0,5 puntos – 4 min.) Indique ¿cuál sería la línea de comando equivalente a este programa?

<u>Pregunta 3</u> (4 puntos – 32 min.) Considere el siguiente código:

```
$ cat -n 2019-2_ex1_q3.c | expand
     1  #include <sys/types.h>
     1  #include <sys/types.h>
     2  #include <sys/wait.h>
     3  #include <stdio.h>
     4  #include <stdlib.h>
     5  #include <unistd.h>
     6
     7  void die(char *);
     8  void close_pipes(void);
     9
    10  int pipefd[3][2];
    11
    12  int
    13  main(void) {
    14      pid_t p1=-1, p2=-1, p3=-1, p4=-1;
    15      int i, r, fd0, fd1;
    16
    17      for (i=0; i<3; i++)
    18          if (pipe(pipefd[i]) < 0) die("pipe() error\n");
    19
    20      r = (p1=fork()) + !(p2=fork()) ? (p3=fork()) : !(p4=fork());
    21
    22      if (p1 && p2 && !p3 && p4==-1) {
    23          dup2(pipefd[2][0],0); close_pipes();
    24          execl("/usr/bin/wc", "wc", "-w", NULL);
    25          die("execl() error\n");
```

```
26      }
27      if ((!p1 && !p4) || (!p2 && !p3)) {
28          fd0 = p1 ? pipefd[1][0] : pipefd[0][0];
29          fd1 = p1 ? pipefd[2][1] : pipefd[1][1];
30          dup2(fd0,0); dup2(fd1,1); close_pipes();
31          if (p1) {
32              execl("/bin/grep", "grep", "vfork", NULL);
33              die("execl() error\n");
34          }
35          if (p2) {
36              execl("/bin/sed", "sed", "s/fork/vfork/g", NULL);
37              die("execl() error\n");
38          }
39      }
40      sleep(1);
41      if (!p1 && !p2) {
42          if (p3) {
43              dup2(pipefd[0][1],1);
44              close_pipes();
45              execl("/bin/cat", "cat", "2019-2_ex1_q3.c", NULL);
46              die("execl() error\n");
47          }
48          close_pipes();
49          execl("/bin/ps", "ps", "-l", NULL);
50          die("execl() error\n");
51      }
52      close_pipes();
53      while (waitpid(-1,NULL,0) > 0);
54      exit(0);
55  }
56
57  void
58  die(char *s) {
59      if (s != (char *)NULL) {
60          while (*s) (void) write(2,s++,1);
61      }
62      exit((s == (char *)NULL)? 0 : 1);
63  }
64
65  void
66  close_pipes(void) {
67      int i;
68      for (i=0; i<3; i++) {
69          if (close(pipefd[i][0]) < 0) die("close() error\n");
70          if (close(pipefd[i][1]) < 0) die("close() error\n");
71      }
72  }
```

Enumere los procesos creados por el programa por los niveles y de la izquierda a la derecha, usando los números a partir de 20: el proceso 20 es el padre, el proceso 21 es el primer hijo, el proceso 22 es el 2do hijo, etc. Presente el árbol de procesos creados, en las aristas indique el número de `fork()` con cual fue creado cada proceso. Con las tuplas (`p1,p2,p3,p4`) al lado de cada proceso indique los valores de estas variables, también el valor de la variable `r`. Si un proceso cambia su programa, indique al cuál programa. Presente las tuberías creadas con sus conexiones finales a los procesos.

<u>Pregunta 4</u> (2 puntos – 16 min.) (*OSIDP7E*) Consider the following program:

```
 A: {                                    B: {
      shared int x;                           shared int x;

1     x = 10;                           1     x = 10;
2     while (1) {                       2     while (1) {
3         x = x - 1;                    3         x = x - 1;
4         x = x + 1;                    4         x = x + 1;
5         if (x != 10)                  5         if (x != 10)
6             printf("x is %d", x);     6             printf("x is %d", x);
      }                                       }
     }                                       }
```

Note that the scheduler in a uniprocessor system would implement pseudo-parallel execution of these two concurrent processes by interleaving their instructions, without restriction on the order of the interleaving.

a) Show a sequence (i.e., trace the sequence of interleaving of statements) such that the statement "`x is 10`" is printed.

b) Show a sequence such that the statement "`x is 8`" is printed. You should remember that the increment/decrements at the source language level are not done atomically.

Pregunta 5 (4 puntos – 32 min.) (*The Little Book of Semaphores – 2nd Ed*) En la última práctica ya vimos un monitor que organizaba una barrera para formar los grupos de 9 visitantes a un museo. Ahora será el caso de organización de una barrera usando semáforos con las operaciones *wait()* y *signal()* como el siguiente código donde el 9no visitante abre la barrera:

```
 1  n = the number of threads > 9
 2  count = 0
 3  mutex = Semaphore (1)
 4  barrier = Semaphore (0)

 5  rendezvous

 6  mutex.wait()
 7  count = count + 1
 8  mutex.signal()

 9  if count == 9: barrier.signal()
10  barrier.wait()

11  critical point
```

a) (1 punto – 8 min.) ¿Pueden pasar la barrera 9 visitantes? ¿Algunos? ¿Ninguno? ¿Todos `n` visitantes? Explique la posibilidad de cada caso.

b) (1 punto – 8 min.) ¿Cómo se responderá a las mismas preguntas si el código se cambia al siguiente?

```
 5  rendezvous

 6  mutex.wait()
 7  count = count + 1

 8  if count == 9: barrier.signal()

 9  barrier.wait()
10  barrier.signal()
11  mutex.signal()

12  critical point
```

c) (2 puntos – 16 min.) Proponga la solución correcta, y explique el funcionamiento de la misma.

Pregunta 6 (3 puntos – 24 min.)

a) (1 punto – 8 min.) El entorno de trabajo Windows 3.1, montado sobre el Sistema Operativo DOS, en esencia monotarea, permitía manejar la idea de procesos, con una planificación no-apropiativo (*non-preemptive*) la cuál suele decirse de características cooperantes entre los procesos, principalmente interactivos, cuyo lazo principal recibe mensajes de una sola cola, para por ejemplo, recibir entrada del teclado, movimiento o clic del mouse, repintado de ventana al pasar al frente, etc. Si el proceso no tiene mensajes o termina de procesar alguno, deja el procesador para que otro proceso realice similar lazo, de acuerdo a su código. ¿Qué sucede si una función de manejo de mensaje toma demasiado procesador o peor aún, "entra en *loop*"? ¿Cómo puede solucionarse?

b) (2 puntos – 16 min.) Suponga que dos procesos, P1 y P2, llegan simultáneamente o están listos para ejecutar su ráfaga en el instante 0. P1 y P2 tienen ráfagas de 50ms. Desarrolle la ejecución de estos procesos en el tiempo (gráfica de Gantt) para un quantum de: i) 25ms; ii) 50ms; iii) 100ms. Calcule: tiempo promedio de espera, y número de cambios de contexto o estado.

Preparado por FS (1,2,6) y VK (3,4,5) con LibreOffice Writer
en Linux Mint 19.2 "Tina"

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

Pando, 16 de octubre de 2019