PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

1ra práctica (tipo a)
(Segundo semestre de 2018)

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.
Puntaje total: 20 puntos

Pregunta 1 (2 puntos – 10 min.) Considere el siguiente código y los resultados de su ejecución. Explique el proceso de su ejecución y los cambios de los valores en las variables. También explique cómo puede suceder que las direcciones de las variables son las mismas mientras que sus valores son diferentes.

```
$ cat -n fork_pointer_v2.c
     1  #include <stdio.h>
     2  #include <stdlib.h>
     3  #include <unistd.h>
     4  #include <sys/wait.h>
     5
     6  int value;
     7
     8  int
     9  main(void) {
    10      int* p= &value;
    11
    12      printf("value = %5d (%p), p = %p, *p = %d\n",value,&value,p,*p);
    13      if (*p= fork()) wait(NULL);
    14      printf("value = %5d (%p), p = %p, *p = %d\n",value,&value,p,*p);
    15      exit(0);
    16  }
$ gcc fork_pointer_v2.c
$ ./a.out
value =     0 (0x55cf19af7014), p = 0x55cf19af7014, *p = 0
value =     0 (0x55cf19af7014), p = 0x55cf19af7014, *p = 0
value = 14495 (0x55cf19af7014), p = 0x55cf19af7014, *p = 14495
```

Pregunta 2 (2 puntos – 10 min.) Complete los resultados de la siguiente sesión en *shell*:

```
vk@kaperna ~/clases/so/progs $ >foo
vk@kaperna ~/clases/so/progs $ ls -il foo
13505139 -rw-rw-r-- 1 vk vk 0 set 12 23:32 foo
vk@kaperna ~/clases/so/progs $ cp foo bar
vk@kaperna ~/clases/so/progs $ ls -il {foo,bar}
13505140 -rw-rw-r-- 1 vk vk 0 set 12 23:33 bar
13505139 -rw-rw-r-- 1 vk vk 0 set 12 23:32 foo
vk@kaperna ~/clases/so/progs $ mv foo .//..///progs/./.baz
vk@kaperna ~/clases/so/progs $ ls -il {foo,bar,baz}
...
```

Pregunta 3 (6 puntos – 30 min.) Considere el siguiente código:

```
$ cat -n 2018-2_pr1.c
     1  #include <stdio.h>
     2  #include <stdlib.h>
     3  #include <unistd.h>
     4  #include <signal.h>
     5  #include <sys/wait.h>
     6
     7  int
     8  main(int argc, char *argv[]) {
```

```
 9      int n, fd[2], p;
10
11      if (argc < 2) {
12          printf("Ingrese un número: "); scanf("%d",&n);
13      } else { n= atoi(argv[1]); }
14      printf("%d ",n);
15      fflush(stdout);
16
17      pipe(fd);
18      p= fork()? fork() : 0;
19      if (p) {
20          write(fd[1],&n,sizeof(int));
21          close(fd[0]); close(fd[1]);
22          while (wait(NULL) != -1);
23          printf("\n");
24          exit(0);
25      } else {
26          if ((getpid() != getppid()+1) && (getpid() != getppid()+2)) {
27              printf("Next time, please!\n");
28              killpg(getpgrp(),SIGKILL);      /* enviar la señal a cada uno de los procesos del grupo */
29          }
30          while (1) {
31              read(fd[0],&n,sizeof(int));
32              if (n == 1) {
33                  break;
34              }
35              if ((getpid() & 1) && (n & 1)) {
36                  n= (n<<1)+n+1;
37                  printf("%d ",n);
38                  fflush(stdout);
39              }
40              if (!(getpid() & 1) && !(n & 1)) {
41                  n>>= 1;
42                  printf("%d ",n);
43                  fflush(stdout);
44              }
45              write(fd[1],&n,sizeof(int));
46          }
47          close(fd[0]); close(fd[1]);
48          exit(0);
49      }
50  }
```

```
$ gcc 2018-2_pr1.c

$ ./a.out 2
2 1 Terminado
```

Después de imprimir dos valores, el programa "se cuelga". Del otro terminal se ingresan las siguientes órdenes que provocan el mensaje "`Terminado`" en el 1er terminal:

```
$ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
vk       10872  0.0  0.0  25676  5584 pts/0    Ss   set12   0:00 bash
vk       26251  0.0  0.0   4508   740 pts/0    S+   21:16   0:00 ./a.out
vk       26253  0.0  0.0   4508    80 pts/0    S+   21:16   0:00 ./a.out
vk       26274  0.1  0.0  25404  5128 pts/2    Ss   21:16   0:00 bash
vk       26287  0.0  0.0  39964  3500 pts/2    R+   21:16   0:00 ps u
$ killall a.out
```

a) (3 puntos – 15 min.) Suponiendo que las líneas 27 y 28 no se ejecutan, explique comportamiento de cada proceso para el caso de ejecución presentado con el estado "colgado".

b) (1 punto – 5 min.) ¿Cómo se corrige el error?

c) (1 punto – 5 min.) Para el programa corregido, ¿cuál será la salida para el dato de entrada 5? ¿Para el dato de entrada 13?

d) (1 punto – 5 min.) Si en el programa corregido eliminar la línea 15, se obtiene la siguiente salida:

```
$ gcc 2018-2_pr1.c
$ ./a.out 2
2 1 2 2
$
```

Explique qué proceso imprime cada valor y por qué.

Pregunta 4 (6 puntos – 30 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) (1 punto – 5 min) Agregue los comentarios de la función *main*. (líneas 62 a 80).

(b) (2 puntos – 10 min) Agregue los comentarios de la función *join*. (líneas 14 a 60).

(c) (2 puntos – 10 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) (1 punto – 5 min.) Indique ¿cuál sería la línea de comando equivalente a este programa ?

Pregunta 5 **(2 puntos – 10 min.)** *(Llamadas al sistema / Interrupciones)* Utilice la llamada al sistema ***read(fd,buf,nbytes)*** para:
(a) (0,5 puntos – 2,5 min.) Describir lo que ocurre en el programa de usuario (lo que comúnmente hace el compilador para llamar una función) y describir lo que ocurre al interior de la "función de librería".

(b) (0,5 puntos – 2,5 min.) Describir lo que ocurre efectivamente para realizar el "*read"* en sí.

(c) (0,5 puntos – 2,5 min.) La llamada al sistema en cuestión, READ, puede no retornar inmediatamente si los datos solicitados no están "listos" para ser pasados al proceso en la dirección dada por ***buf.*** Suponiendo que el código ejecutado en un lazo de lectura, para evaluar si se continúa en el lazo, hasta el momento en que se determina que no están listos, tarda 200 ns, que estarán listos recién 10 ms después, y que usar los datos tarda otros 200ns, para volver a evaluar el lazo , haga una gráfica que muestre la ejecución repetida de ***read(fd,buf,nbytes)*** las repeticiones que crea convenientes para llegar a unos 100ms, y calcule el porcentaje de tiempo que la CPU está ocupada.

(d) (0,5 puntos – 2,5 min.). Bajo las condiciones de procesamiento en **(c)**, ¿cuántos procesos que hagan lo mismo se necesitarían para que la CPU esté cercana al 100% de uso ?

Pregunta 6 **(2 puntos – 10 min.)** *(Estructuras de Sist. Operativos)* La estructura **monolítica** es la más común en muchos sistemas operativos, mientras que la estructura de ***microkernel*** tiene características, digamos, deseable. Describa muy concretamente la estructura de **microkernel** (1.0 punto) y señale alguna ventaja y una desventaja (1 punto)

La práctica ha sido preparada por FS (4-6) y VK (1-3)
con LibreOffice Writer en Linux Mint 19 Tara.

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

Pando, 14 de septiembre de 2018