PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

2da práctica (tipo a)
(Primer semestre de 2022)

Horarios 0781, 0782: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **La presentación, la ortografía y la gramática influirán en la calificación.**
Puntaje total: 20 puntos

Pregunta 1 (6 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_078X_P2_P1_Buzón** de la **Práctica 2** en PAIDEIA **antes de las 09:40**. Por cada 3 minutos de retardo son -2 puntos.
El nombre de su archivo debe ser <*su_código_de_8_dígitos*>`_21.txt`. Por ejemplo, `20171903_21.txt`.

El programa `foo.c`:


```
10 int
11 main(int argc, char **argv)
12 {
13     int j;
14
15     printf("Process %d (child of %d) executing %s:\n",
16         getpid(), getppid(), argv[0]);
17     for (j = 0; j < argc; j++)
18         printf("\targv[%d]: %s\n", j, argv[j]);
19
20     exit(EXIT_SUCCESS);
21 }
```

```
$ gcc foo.c -o foo
$ ./foo qux quux quuz
Process 102810 (child of 98093) executing ...:
...
```

a) (1 punto) Complete todo lo marcado con tres puntos rojos ("...") en las líneas anteriores.

El programa `bar.c`:

```
10 int
11 main(int argc, char **argv)
12 {
13     char *newargv[] = { NULL, "hello", "world", NULL };
14     char *newenviron[] = { NULL };
15
16     if (argc != 2) {
17         fprintf(stderr, "Usage: %s <file>\n", argv[0]);
18         exit(EXIT_FAILURE);
19     }
20
21     printf("Process %d (child of %d) executing %s with %s\n", getpid(),
22         getppid(), argv[0], argv[1]);
23
24     newargv[0] = argv[1];
25
26     execve(argv[1], newargv, newenviron);
27     perror("execve");
28     exit(EXIT_FAILURE);
29 }
```

```
$ gcc bar.c -o bar
$ ./bar ./foo
Process 105540 (child of 98093) executing ... with ...
Process ... (child of ...) executing ...:
...
```

b) (2 puntos) Complete todo lo marcado con tres puntos rojos ("...") en las líneas anteriores.

```
$ ./bar /bin/echo
Process 106775 (child of 98093) executing ... with ...
...
```

c) (1 punto) Complete todo lo marcado con tres puntos rojos ("...") en las líneas anteriores.

```
$ cat > baz
#!./foo baz-arg
^D
$
$ ./bar baz
Process 107621 (child of 98093) executing ... with ...
execve: Permission denied
$ ./bar ./baz
Process 107630 (child of 98093) executing ... with ...
execve: Permission denied
```

d) (1 punto) No se necesita completar la salida en las líneas anteriores sino **indicar la razón** del error sucedido.

Después de resolver el problema, la ejecución del programa `bar` sucede sin problemas:

```
$ ./bar baz
Process 111411 (child of 98093) executing ... with ...
Process ... (child of ...) executing ...:
...
```

e) (1 punto) ¿Podrá presentar la salida proporcionada por la ejecución del programa completando lo marcado con "..."?

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 20.3 "Una"

Profesor del curso: V. Khlebnikov

Lima, 6 de mayo de 2022

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

2da práctica (tipo a)
(Primer semestre de 2022)

Horarios 0781, 0782: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **La presentación, la ortografía y la gramática influirán en la calificación.**
Puntaje total: 20 puntos

Pregunta 2 (6 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_078X_P2_P2_Buzón** de la **Práctica 2** en PAIDEIA **antes de las 10:20**. Por cada 3 minutos de retardo son -2 puntos.
El nombre de su archivo debe ser <*su_código_de_8_dígitos*>_22.txt. Por ejemplo, 20171903_22.txt.

```
$ cat -n childDataEchange.c
     1   #include <stdio.h>
     2   #include <unistd.h>
     3   #include <stdlib.h>
     4   #include <sys/wait.h>
     5
     6   /*
     7   Este programa crea dos abanicos de procesos. Es decir, el padre crea
     8   dos hijos y cada uno de ellos crea un abanico de n procesos.
     9   Ejm 2.6 del libro UNIX Programacion Practica - Kay Robbins
    10                                                 Steve Robbins
    11   Modificado por Alejandro Bello Ruiz - Informática PUCP
    12   Moificado por vk
    13   */
    14
    15   int
    16   main(int argc,char **argv)
    17   {
    18       int i,j,n,*ptr;
    19       int w,r;
    20       pid_t pid,pidOne;
    21       char command[30];
    22
    23       n = atoi(argv[1]);
    24
    25       ptr = (int *) calloc(...,sizeof(int));
    26       for (i=0; i<...; i+=2) pipe(ptr+i);
    27
    28       pidOne = getpid();
    29
    30       for (i=0; i<2; i++)
    31           if (!fork()) {
    32               for (j=0; j<n; j++)
    33                   if (!fork()) {
    34                       int sn,rn;
    35
    36                       pid = getpid();
    37                       srand(pid);
    38                       sn = rand()%1000;
    39                       fprintf(stderr,
    40                           "i,j=%d,%d; pid=%d, random number=%d\n",
    41                           i,j,pid,sn);
    42                       sleep(1);
    43
    44                       fprintf(stderr, "%d to exchange...\n", pid);
    45                       w = ...;
    46                       r = ...;
    47                       write(ptr[w],&sn,sizeof(int));
    48                       read(ptr[r],&rn,sizeof(int));
    49                       fprintf(stderr,
    50                           "i,j=%d,%d; pid=%d, sent number=%d, received number=%d\n",
    51                           i,j,pid,sn,rn);
    52
    53                       if (i==1 && j==n-1) {
    54                           sprintf(command,"...",pidOne);
    55                           system(command);
```

```
    56                      }
    57                      pause();
    58                  }
    59              wait(NULL);
    60          }
    61      wait(NULL);
    62      return 0;
    63  }
```

```
$ gcc childDataEchange.c -o childDataEchange

$ ln childDataEchange cde

$ ./cde 2
i,j=0,0; pid=122266, random number=333
i,j=1,0; pid=122267, random number=112
i,j=0,1; pid=122268, random number=921
i,j=1,1; pid=122269, random number=229
122266 to exchange...
122267 to exchange...
i,j=1,0; pid=122267, sent number=112, received number=333
i,j=0,0; pid=122266, sent number=333, received number=112
122268 to exchange...
122269 to exchange...
i,j=1,1; pid=122269, sent number=229, received number=921
i,j=0,1; pid=122268, sent number=921, received number=229
systemd(1)──gnome-terminal-(63423)──bash(122001)──cde(122263)─┬─cde(122264)─┬─cde(122266)
                                                              │             └─cde(122268)
                                                              └─cde(122265)─┬─cde(122267)
                                                                            └─cde(122269)──sh(122270)──pstree(122271)
^C
$
```

a) (1 punto) Complete las líneas 25 y 26 del código del programa. La variable `ptr` se usa en las líneas 47 y 48.

b) (4 puntos) Complete las líneas 45 **(2 puntos)** y 46 **(2 puntos)** del código del programa. Las variables `w` y `r` se usan en las líneas 47 y 48.

c) (1 punto) Complete la línea 54.

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 20.3 “Una”

Profesor del curso: V. Khlebnikov

Lima, 6 de mayo de 2022

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
2da práctica (tipo a)
(Primer semestre de 2022)

Horarios 0781, 0782: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **La presentación, la ortografía y la gramática influirán en la calificación.**
Puntaje total: 20 puntos

Pregunta 3 (8 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_078X_P2_P3_Buzón** de la **Práctica 2** en PAIDEIA **antes de las 11:00**. Por cada 3 minutos de retardo son -2 puntos.
El nombre de su archivo debe ser <*su_código_de_8_dígitos*>`_23.txt`. Por ejemplo, `20171903_23.txt`.

a) (2 puntos) Consideremos un programa simple que trabaja con una cuenta bancaria:

```
// Package bank implements a bank with only one account.
package bank

var balance int

func Deposit(amount int) { balance += amount }

func Balance() int { return balance }
```

donde una función realiza un deposito a la cuenta y la otra reporta el monto total en la cuenta.

Ahora imaginemos que dos personas, Alice y Bob, realizan dos transacciones simultaneas a una cuenta mancomunada (o sea, asociada a 2 o más personas) ejecutando al mismo tiempo los siguientes códigos:

```
// Alice:
go func() {
    bank.Deposit(200)                   // A1
    fmt.Println("=", bank.Balance())    // A2
}()


// Bob:
go bank.Deposit(100)                    // B
```

o sea, Alice deposita $200 y verifica el balance, y Bob deposita $100.

Como la ejecución es concurrente, entonces el orden de ejecución de los pasos A1 y A2 con B no se puede predecir. Entre los posibles hay los siguientes:

```
Alice primero           Bob primero        Alice/Bob/Alice
            0                     0                      0
 A1       200           B       100           A1       200
 A2    "= 200"          A1      300           B        300
 B        300           A2   "= 300"          A2    "= 300"
```

Usted sabe qué es *race condition*. ¿Este programa presentado lo contiene? ¿Cómo usted lo puede justificar?

b) (2 puntos) Considere las siguientes dos funciones que se ejecutan concurréntemente accediendo a dos variables compartidas:

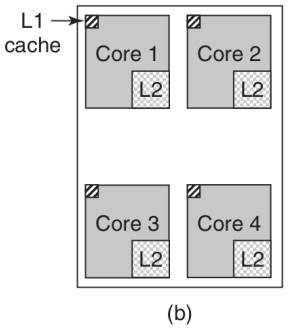
```
var x, y int

go func() {
    x = 1                       // A1
    fmt.Print("y:", y, " ")     // A2
}()

go func() {
    y = 1                       // B1
    fmt.Print("x:", x, " ")     // B2
}()
```

En este caso el programa no será determinístico y se puede esperar los siguientes resultados:

```
    y:0 x:1
    x:0 y:1
    x:1 y:1
    y:1 x:1         A1,B1,A2,B2 o B1,A1,A2,B2
```



(b)

Pero para su sorpresa se puede obtener estos dos resultados:

```
    x:0 y:0
    y:0 x:0
```

si las funciones se ejecutan en los procesadores (o sus núcleos) diferentes. Explique cómo es posible esto usando la siguiente Figura 1.8(b) del libro MOS4E:

c) (4 puntos)

c1) ¿Puede un hilo (*thread*) adquirir más de un cerrojo (de tipo *mutex*)?

c2) Si `semaphore mutex=1;` y se ejecutan muchas operaciones `down(&mutex);` , ¿qué pasará?

c3) Si `semaphore mutex=1;` y un hilo (*thread*) ejecuta `down(&mutex); down(&mutex);` , ¿qué pasará?

c4) ¿Es necesario que un hilo (*thread*) siempre debe bloquearse si el recurso no está disponible? Explique.

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 20.3 “Una”

Profesor del curso: V. Khlebnikov

Lima, 6 de mayo de 2022