PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

1ra práctica (tipo a)
(Primer semestre de 2021)

Horario 0781: 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_0781_P1_P1** de la **Práctica 1** 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*>`_11.txt`. Por ejemplo, `20171903_11.txt`.

En el Laboratorio 1 de este curso fue presentado el programa `binarytree.c`:

```
1  /* ************************************************************/
2  /* binarytree.c (c) 2010 Alejandro T. Bello Ruiz, GPL-like     */
3  /* ********** ************************************************/
4  #include <stdio.h>
5  #include <unistd.h>
6  #include <math.h>
7  #include <stdlib.h>
8  #include <string.h>
9  #include <sys/wait.h>
10
11 double final;
12
13 void crea_arbol(int);
14
15 int main(int narg, char *argv[])
16 {  int n;
17
18    n=atoi(argv[1]);
19    final=pow(2,(n-1));
20    crea_arbol(1);
21    return 0;
22 }
23
24 void crea_arbol(int x)
25 {  char cadena[60];
26
27    sprintf(cadena,"Soy el proceso %d con pid %d y ppid %d\n",x,getpid(),getppid());
28    write(1,cadena,strlen(cadena));
29    if(x >= final) return;
30    if(!fork()) { crea_arbol(2*x); exit(0); }
31    if(!fork()) { crea_arbol(2*x+1);exit(0);}
32    wait(NULL);
33    wait(NULL);
34 }
35
```

(a) (2 puntos) Considerando que en la variable `x`, el argumento de la función `crea_arbol`, se entrega el **número consecutivo** del proceso (1, 2, 3, 4, 5, …), el que se imprime en la línea 27, ¿cuál es el significado del valor que se calcula para la variable `final` en la línea 19 y el que se usa en la línea 29 definiendo la condición de salida? Explique este significado en términos del **número consecutivo** del proceso.

(b) (2 puntos) Si se pretende modificar el programa `binarytree.c` para obtener el programa `ternarytree.c` que construye un árbol de procesos donde cada nodo del árbol tiene 3 hijos, ¿cuáles serían sus modificaciones? Indique **solamente** las líneas con sus modificaciones.

(c) (2 puntos) Si ejecutar el programa `ternarytree.c` para crear los árboles de 2, de 3 y de 4 niveles, ¿cuáles serán las secuencias de los números consecutivos de los procesos creados? Coloque los números por niveles.

Pregunta 2 (6 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_0781_P1_P2** de la **Práctica 1** 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*>`_12.txt`. Por ejemplo, `20171903_12.txt`.

En el Laboratorio 1 de este curso fue presentado el programa `binarytree1.c`:

```c
/* ************************************************************/
/* binarytree.c (c) 2021 Alejandro T. Bello Ruiz, GPL-like     */
/* ********** ********************************************** */
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>

double final;

void crea_arbol(int);

int main(int narg, char *argv[])
{
   final=atoi(argv[1]);
   crea_arbol(0);
   return 0;
}

void crea_arbol(int nivel)
{  int k;

   nivel++;
   for(k=0;k<2;k++)
      if(!fork()) {
         if(nivel < final) crea_arbol(nivel);
         else pause();
      }
   wait(NULL);
   wait(NULL);

}

```

con el objetivo de modificar este programa para que la orden `pstree` sea lanzado desde el mismo programa. El que debe lanzarlo debe ser el último proceso creado, para asegurarse que en la salida se encuentran todos los procesos. Sin usar *pipes*. Sin usar la función de librería `sleep`, pues los procesos ya se encuentran bloqueados.

Razonando, hay que reconocer que el único dato que recibe la función `crea_arbol` es el nivel del árbol y esto no es suficiente para identificar el último proceso que será creado. Hay que proporcionarle más información. Entonces el prototipo de la función será

```
void crea_arbol(int, int);
```

Ahora llegó el tiempo de poner a las variables los nombres más adecuados. El argumento 1 que recibimos después de invocar al programa es la cantidad de niveles que debe tener el árbol construido decrementada en 1. Entonces colocaremos `argv[1]` en la variable `nivel`, y en la variable `final` colocaremos la cantidad de nodos que tiene un árbol binario perfecto (y es nuestro caso) con la cantidad de niveles solicitados. Esto nos dará el número consecutivo del último nodo.

La función `crea_arbol` no cambia su estructura: el mismo bucle `for` de 2 iteraciones; el proceso, que será padre, ejecuta 2 veces `fork` y se bloquea esperando la terminación de ambos hijos; los procesos hijos, si ellos todavía no están en el nivel solicitado, deben crear sus propios hijos del siguiente nivel.

Los cambios vienen en la parte `else` antes de `pause()`. Aquí llegan los procesos del último nivel. Ahora hay que identificar el último de ellos. El 2do argumento de la función `crea_arbol` es un contador que contiene el número consecutivo del proceso. Si este contador coincide con el valor de la variable `final`, este proceso es el último proceso y él debe preparar la orden para *shell* en la función `system` con el número del proceso que es la raíz de todo el árbol (el último proceso lo debe saber) y otra vez invocar a la función `system` para "matar" a todos los procesos que ejecutan este programa.

Naturalmente, el contador que se usa como el 2do argumento de la función `crea_arbol` hay que mantener correctamente en cada nivel.

a) (1 punto) ¿El valor de qué expresión se asigna a la variable `final`?

b) (1 punto) ¿Cuál será la llamada inicial a la función `crea_arbol`?

c) (2 puntos) ¿Cuál será la llamada a la función `crea_arbol` que hacen los procesos que no están en el último nivel? Suponemos que los parámetros formales son los siguientes: `void crea_arbol(int n, int cont)` y el nivel ya está incrementado.

d) (2 puntos) Complete el siguiente código:

```
if (cont == final) {
    char orden[20];

    ...
    ...
    ...
}
pause();
```

Pregunta 3 (8 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_0781_P1_P3** de la **Práctica 1** 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*>`_13.txt`. Por ejemplo, `20171903_13.txt`.

a) (4 puntos) En el Material para el Laboratorio 1 se presenta el programa `anillo.c`:

```c
/*****************************************************************************/
/*            Programa anillo.c                                              */
/*            Tomado del libro: UNIX SYSTEMS Programming                     */
/*            Autores : Kay A. Robbins y Steven Robbins                      */
/*****************************************************************************/
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
int main(int argc,  char *argv[ ]) {
   pid_t childpid;              /* indicates process should spawn another     */
   int error;                   /* return value from dup2 call                */
   int fd[2];                   /* file descriptors returned by pipe          */
   int i;                       /* number of this process (starting with 1)   */
   int nprocs;                  /* total number of processes in ring          */
         /* check command line for a valid number of processes to generate */
   if ( (argc != 2) || ((nprocs = atoi (argv[1])) <= 0) ) {
       fprintf (stderr, "Usage: %s nprocs\n", argv[0]);
       return 1;
   }
   if (pipe (fd) == -1) {       /* connect std input to std output via a pipe */
      perror("Failed to create starting pipe");
      return 1;
   }
   if ((dup2(fd[0], STDIN_FILENO) == -1) ||
       (dup2(fd[1], STDOUT_FILENO) == -1)) {
      perror("Failed to connect pipe");
      return 1;
   }
   if ((close(fd[0]) == -1) || (close(fd[1]) == -1)) {
      perror("Failed to close extra descriptors");
      return 1;
   }
   for (i = 1; i < nprocs;  i++) {         /* create the remaining processes */
      if (pipe (fd) == -1) {
         fprintf(stderr, "[%ld]:failed to create pipe %d: %s\n",
                (long)getpid(), i, strerror(errno));
         return 1;
      }
      if ((childpid = fork()) == -1) {
         fprintf(stderr, "[%ld]:failed to create child %d: %s\n",
                 (long)getpid(), i, strerror(errno));
         return 1;
      }
      if (childpid > 0)               /* for parent process, reassign stdout */
          error = dup2(fd[1], STDOUT_FILENO);
      else                             /* for child process, reassign stdin */
          error = dup2(fd[0], STDIN_FILENO);
      if (error == -1) {
         fprintf(stderr, "[%ld]:failed to dup pipes for iteration %d: %s\n",
                 (long)getpid(), i, strerror(errno));
         return 1;
      }
      if ((close(fd[0]) == -1) || (close(fd[1]) == -1)) {
         fprintf(stderr, "[%ld]:failed to close extra descriptors %d: %s\n",
                (long)getpid(), i, strerror(errno));
         return 1;
      }
      if (childpid)
         break;
   }                                              /* say hello to the world */
   fprintf(stderr, "This is process %d with ID %ld and parent id %ld\n",
           i, (long)getpid(), (long)getppid());
   return 0;
}
```

que forma un anillo de procesos conectados por tuberías.

El proceso que ejecuta este programa, en un bucle `for` crea los procesos restantes. La cantidad total de procesos se indica como el `argv[1]` en el mandato con cual se invoca el programa.

El proceso que ejecuta este bucle `for` primero crea una tubería (línea 36) y después crea un proceso hijo (línea 41). Hay que decir que según el manual de `fork`:

```
DESCRIPTION
       fork() creates a new process by duplicating the calling process. The
       new process is referred to as the child process. The calling process
       is referred to as the parent process.

       The child process and the parent process run in separate memory spaces.
       At the time of fork() both memory spaces have the same content.
...
       *  The child inherits copies of the parent's set of open file descrip-
          tors.  Each file descriptor in the child refers to the same open
          file description (see open(2)) as the corresponding file descriptor
          in the parent. ...
```

Esto significa que el proceso hijo hereda TODOS los descriptores de archivos abiertos por el proceso padre, incluyendo la entrada estándar y la salida estándar.

Regresando al bucle `for`, el proceso padre asigna su salida estándar a la tubería creada y el proceso hijo, heredando los descriptores de la tubería en la variable `fd`, asigna su entrada estándar a la tubería creada. Después ellos cierran sus accesos a la tubería creada por los descriptores de la variable `fd` y el proceso padre abandona el bucle `for`, mientras que el proceso hijo seguirá con la siguiente iteración.

a1) (2 puntos) Cuando el proceso, creado por *shell*, inicia la ejecución de este programa y, por primera vez, entra en el bucle `for` creando al primer proceso hijo (ahora existen solo 2 procesos que ejecutan este programa) y ambos terminan la 1ra iteración (y la única para el proceso padre), ¿a qué está asignada la entrada estándar del padre y a qué está asignada la salida estándar del proceso hijo? Explique su respuesta.

a2) (2 puntos) Si ya existen más de 2 procesos que ejecutan este programa, ¿a qué está asignada la salida estándar del último proceso hijo creado y por qué?

b) (4 puntos) En el Laboratorio 1 se solicitó el desarrollo de un simple programa del cálculo de *n*-ésimo número de Fibonacci a partir del programa proporcionado `anillo.c` para obtener el siguiente resultado (en este caso en 18 milisegundos y creando un anillo de 77 procesos):

```
$ time ./anilloFib 78
El fibonacci de 78 es 8944394323791464

real0m0,017s
user0m0,001s
sys 0m0,000s
$
```

Complete la parte faltante de este código:

```
    if (fpid == getpid()) {
        long a,b,fib;
        char cadena[250];

        sprintf(cadena,"1 1");
        write(STDOUT_FILENO,cadena,sizeof(cadena));
        read(STDIN_FILENO,cadena,sizeof(cadena));
        sscanf(cadena,"%ld %ld",&a,&b);
        fib=a;
        fprintf(stderr,"El fibonacci de %d es %ld\n",nprocs,fib);
    }
    else {
        long a,b,c;
        char cadena[250];

        ...
        ...
        ...
        ...
```

```
        ...
    }
    return 0;
}
```

La práctica ha sido preparada por VK
(usando los programas elaborados por el prof. Alejandro Bello)
con LibreOffice Writer en Linux Mint 20.1 “Ulyssa”

Profesores del curso: V. Khlebnikov

Lima, 30 de abril de 2021