PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
1ra práctica (tipo a)
(Primer semestre de 2016)

Horario 0781: prof. V. Khlebnikov
Horario 0782: prof. A. Bello R.

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 (4 puntos – 20 min.) (*F. Pérez*) Analice **completamente** la siguiente figura de una traza de ejecución de procesos:

(a) (1 punto – 5 min.) El primer detalle: se puede observar el uso de 3 tipos diferentes de líneas dibujadas. Explíquelo.

Utilizaremos un ejemplo muy simplificado del código de un hipotético manejador de un dispositivo de entrada, para un sistema operativo con **multiprogramación**, que opera en modo carácter y usa interrupciones. Téngase en cuenta que en este punto sólo se muestra un esquema simplificado, sin entrar en detalles de aspectos tales como la salvaguarda y recuperación del estado de la ejecución de un programa. A continuación, se muestra el programa 1, que constituye una primera versión del manejador planteado como ejemplo:

```
char *dir_buf; // guarda la dirección donde se copiará el dato leído
// Función que realiza la lectura de un carácter copiándolo a "dir"
int lectura(char *dir) {
    dir_buf = dir;
    out(R_CONTROL, LECTURA); // programa el dispositivo
    Retorna cediendo el control a otro proceso;
}
// rutina de interrupción del dispositivo
void interrupcion() {
    *(dir_buf) = in(R_DATOS); // lee el carácter y lo copia
    Marca que el proceso lector ya puede ejecutar;
    Si dicho proceso es más prioritario, retorna al mismo;
    En caso contrario retorna al proceso interrumpido;
}
```

Esquema planteado parece válido a primera vista. Sin embargo, esta solución no es correcta. El segundo detalle está relacionado con uso de memoria y debe corregirse con la segunda versión del manejador del dispositivo:

```
char buf; // guarda el dato leído
// Función que realiza la lectura de un carácter copiándolo a "dir"
int lectura(char *dir) {
    out(R_CONTROL, LECTURA); // programa el dispositivo
    Guarda estado en ese punto y cede control a otro proceso;
    *dir = buf; // continuará ejecutando esta sentencia
}
// rutina de interrupción del dispositivo
void interrupcion() {
    buf = in(R_DATOS); // lee el carácter y lo copia
    Marca que el proceso lector ya puede ejecutar;
    Si dicho proceso es más prioritario, cede control al mismo;
    En caso contrario retorna al proceso interrumpido;
}
```

Con la traza de ejecución correcta:

(b) (3 puntos – 15 min.) ¿Por qué la primera versión del programa no fue correcta? Téngase en cuenta que cada vez que se cambia de proceso se instala el mapa de memoria del nuevo proceso para que éste continúe su ejecución de forma transparente.

Pregunta 2 (6 puntos – 30 min.) (*S. Mishra*(?)) Analice los siguientes códigos:

```
$ cat -n m.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  int
     8  main()
     9  {
    10    int max = 100, i, c1, c2;
    11    FILE *fd;
    12    int mcpipe1[2], mcpipe2[2], status;
    13    char mc0[10], mc1[10], cc0[10], cc1[10];
    14
    15    if (pipe(mcpipe1)) exit(1);
    16    if (pipe(mcpipe2)) exit(2);
    17    sprintf(mc0, "%d", mcpipe1[0]);
    18    sprintf(mc1, "%d", mcpipe1[1]);
    19    sprintf(cc0, "%d", mcpipe2[0]);
    20    sprintf(cc1, "%d", mcpipe2[1]);
    21    if ((c1 = fork()) == -1) exit(3);
    22    else if (c1 == 0) {
    23      execl("s", "s", mc0, mc1, cc0, cc1, NULL);
    24      exit(4);
    25    }
    26    else {
    27      if ((c2 = fork()) == -1) exit(5);
    28      else if (c2 == 0) {
    29        execl("c", "c", cc0, cc1, NULL);
    30        exit(6);
    31      }
    32    }
    33    close(mcpipe1[0]); close(mcpipe2[0]);
    34    fd = fopen("whole", "w");
    35    for (i = 0; i < max; i++) {
    36      fprintf(fd, "%d\n", i);
    37      write(mcpipe1[1], (void *)&i, sizeof(int));
    38      write(mcpipe2[1], (void *)&i, sizeof(int));
    39    }
    40    fclose(fd);
    41    wait(&status);
    42    wait(&status);
    43  }

$ cat -n s.c
     1  #include <stdio.h>
     2  #include <sys/types.h>
     3  #include <sys/wait.h>
     4  #include <unistd.h>
     5  #include <stdlib.h>
     6
     7  int
     8  main(int argc, char *argv[])
     9  {
    10    int i, max = 100;
    11    FILE *fd;
    12    int mcpipe1[2], mcpipe2[2], num;
    13
    14    mcpipe1[0] = atoi(argv[1]);
    15    mcpipe1[1] = atoi(argv[2]);
    16    mcpipe2[0] = atoi(argv[3]);
    17    mcpipe2[1] = atoi(argv[4]);
    18    close(mcpipe1[1]); close(mcpipe2[0]); close(mcpipe2[1]);
    19    fd = fopen("ss", "w");
    20    for (i = 0; i < max; i++) {
    21      read(mcpipe1[0], (char *)&num, sizeof(int));
    22      fprintf(fd, "%d\n", num * num);
    23    }
    24    fclose(fd);
    25    exit(0);
    26  }

$ cat -n c.c
     1  #include <stdio.h>
     2  #include <unistd.h>
     3  #include <stdlib.h>
     4
     5  int
     6  main(int argc, char *argv[])
```

```
 7  {
 8    int i, max = 100;
 9    FILE *fd;
10    int mcpipe2[2], num;
11
12    mcpipe2[0] = atoi(argv[1]);
13    mcpipe2[1] = atoi(argv[2]);
14
15    fd = fopen("cs", "w");
16    for (i = 0; i < max; i++) {
17      read(mcpipe2[0], (char *)&num, sizeof(int));
18      fprintf(fd, "%d\n", num*num*num);
19    }
20    fclose(fd);
21    exit(0);
22  }
```

(a) (2 puntos – 10 min.) Explique funcionamiento de cada proceso.

(b) (1 punto – 5 min.) Si el proceso hijo hereda del proceso padre los descriptores de acceso a las tuberías (creadas por el proceso padre antes de la creación del proceso hijo), entonces ¿qué sentido tiene pasar estos descriptores como parámetros?

(c) (3 puntos – 15 min.) El `man 2 read` dice "`RETURN VALUE: On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number.`" Pero si reemplazar las líneas 16-19 en el código `c.c` por

```
16    while (read(mcpipe2[0], (char *)&num, sizeof(int)))
17      fprintf(fd, "%d\n", num*num*num);
```

el programa "se cuelga". ¿Por qué **(1 punto)**? ¿Cómo arreglar que `while` funcione? (**2 puntos** si no se olvida nada.)

<u>Pregunta 3</u> (10 puntos – 50 min.) Analice el siguiente código:

```
$ cat -n e.c
1   #include <errno.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <string.h>
5   #include <unistd.h>
6   #include <sys/types.h>
7   #include <sys/wait.h>
8
9   #define N 15
10  #define BOUND 51
11
12  int main(void) {
13  pid_t childpid;     /* indicates process should spawn another     */
14  int error;          /* return value from dup2 call                */
15  int fd[2];          /* file descriptors returned by pipe          */
16  int i;              /* number of this process (starting with 1)   */
17  int nprocs=N;       /* total number of processes in ring          */
18  int dato,k;
19  int flag=-1;
20
21  if (pipe (fd) == -1) {      /* connect std input to std output via a pipe */
22      perror("Failed to create starting pipe");
23      return 1;
24  }

25  if ((dup2(fd[0], STDIN_FILENO) == -1) ||(dup2(fd[1], STDOUT_FILENO) == -1)) {
26      perror("Failed to connect pipe");
27      return 1;
28  }
29  if ((close(fd[0]) == -1) || (close(fd[1]) == -1)) {
30      perror("Failed to close extra descriptors");
31      return 1;
32  }
33  for (i = 1; i < nprocs;  i++) {         /* create the remaining processes */
34      if (pipe (fd) == -1) {
35         fprintf(stderr, "[%ld]:failed to create pipe %d: %s\n", (long)getpid(), i, strerror(errno));
36         return 1;
37      }
38      if ((childpid = fork()) == -1) {
39         fprintf(stderr, "[%ld]:failed to create child %d: %s\n", (long)getpid(), i, strerror(errno));
40         return 1;
41      }
42      if (childpid > 0)               /* for parent process, reassign stdout */
43          error = dup2(fd[1], STDOUT_FILENO);
```

```
44      else                                /* for child process, reassign stdin */
45          error = dup2(fd[0], STDIN_FILENO);
46      if (error == -1) {
47         fprintf(stderr, "[%ld]:failed to dup pipes for iteration %d: %s\n",
                               (long)getpid(), i, strerror(errno));
48         return 1;
49      }
50      if ((close(fd[0]) == -1) || (close(fd[1]) == -1)) {
51         fprintf(stderr, "[%ld]:failed to close extra descriptors %d: %s\n",
                               (long)getpid(), i, strerror(errno));
52         return 1;
53      }
54      if (childpid) break;
55   }
56   if(i==1) {
57         for(k=2;k<BOUND;k++) write(STDOUT_FILENO,&k,sizeof(int));
58         write(STDOUT_FILENO,&flag,sizeof(int));
59         read(STDIN_FILENO,&dato,sizeof(int));
60         fprintf(stderr,"%d\n",dato);
61    } else {
62         int t;
63         read(STDIN_FILENO,&dato,sizeof(int));
64         fprintf(stderr,"%d ",dato);
65         while(dato != -1) {
66            read(STDIN_FILENO,&t,sizeof(int));
67            if((t % dato) == 0) continue;
68            write(STDOUT_FILENO,&t,sizeof(int));
69         }
70         write(STDOUT_FILENO,&flag,sizeof(int));
71   }
72   return 0;
73 }
```

a) (3 puntos – 15 minutos) ¿Cuál es la salida del programa? (Emplee una sola línea.)

b) (3 puntos – 15 minutos) Explique cómo logra obtener su objetivo el programa, lleve a cabo su explicción sólo haciendo referencia a los procesos padre e hijos, no haga uso de las instrucciones del programa ni de sus variables.

c) (4 puntos – 20 minutos) Si se elimina la línea 54, **(1 punto)** ¿Cuántos procesos se crean? **(2 puntos)** ¿Se rompe la comunicación entre los procesos, haga un esquema (incluyendo *pipes*) para N = 4? **(1 punto)** ¿Qué cambios hay que hacer para que el programa aún obtenga el mismo resultado?

La práctica ha sido preparada por AB (3) y VK(1,2)
con LibreOffice Writer en Linux Mint 17.3 Rosa.

Profesores del curso: (0781) V. Khlebnikov
(0782) A. Bello R.

Pando, 8 de abril de 2016