Trabajo Práctico N.º1
Gestión de Procesos

Titular: Ing. Rubén L.M.Castaño
Jefe T.P.: Ing. Roberto A. Miño
Aydante de 1ra.: Lic. Claudio O. Biale

Ejercicios Analíticos:

1) Desarrolle:

a) ¿Cuál es la salida del siguiente programa?. Considere que las llamadas no fallan.

```
int main(void) {
      pid_t pid = fork();

      if (pid == 0) {
            printf("Sistemas\n");
            exit(0);
      }
      wait(NULL);
      printf("Operativos\n");
      exit(0);
}
```

b) ¿Cuál es la salida del programa del punto 1.a si se quita la primer sentencia exit(0)?.

c) ¿Cuál es la salida del siguiente programa?. Considere que las llamadas no fallan.

```
int main (void){
      int i;
      pid_t pid1, pid2;

      for(i = 0; i < 2; i++){
            pid1 = fork();
            pid2 = fork();
            if (pid1 > 0) {
                  printf("1\n");
            }
            if (pid2 == 0) {
                  printf("2\n");
            }
      }
      exit(0);
}
```

d) Dibuje un esquema que muestre la jerarquía de procesos resultante de la ejecución del programa del punto 1.c.

2) Dado el siguiente programa, denominado *cp1*:

```
int main(int argc, char * argv[]) {
      int i, cant;
      cant = atoi(argv[1]);
      for (i = 0; i <= cant; i++) {
            fork();
            printf("%d\n", i);
```

```
        }
}
```

Desarrolle:

a) Considerando que las llamadas no fallan, dibuje un esquema que muestre la jerarquía de procesos si el usuario ejecuta:

./cp1 3

b) Escriba el código necesario para que todos los procesos que se crean cuando se invoca a *fork()* tengan un mismo padre, considere que cada hijo debe imprimir el valor de la variable *i* y terminar. Indique con (*) la ubicación de dicho código.

c) Dada la modificación realizada en el punto 2.b, indique cuántos procesos hijos se crean si el usuario ejecuta:

./cp1 4

d) Tomando en cuenta la modificación realizada en el punto 2.b, escriba el código necesario para que el proceso padre luego de haber creado a todos los procesos hijos espere por ellos e imprima que proceso va terminando. Indique con (*) la ubicación de dicho código.

3) Dado el siguiente programa, denominado *cp1*:

```
int main (int argc, char *argv[]) {
      while(1){
            execlp ("kcalc", "kcalc", NULL);
            printf ("Ap. 1 ejecutada\n");
            execlp ("xload", "xload", NULL);
            printf ("Ap. 2 ejecutada\n");
            return 0;
      }
}
```

a) Indique que ocurre al ejecutar el programa. Considere que las llamadas no fallan.

b) Realice las modificaciones necesarias para que el programa lance una sola vez de manera concurrente los aplicativos *kcalc* y *xload*.

c) Tomando en cuenta la modificación realizada en el punto 1.b, escriba el código necesario para que el proceso padre luego de haber realizado el lanzamiento de manera concurrente de *kcalc* y *xload* espere por su finalización e imprima que proceso va terminando. Indique con (*) la ubicación de dicho código.

d) Si el programa *xload* crea dos procesos hijos, dibuje un esquema que muestre la jerarquía de procesos si el usuario ejecuta *cp1*.

4) Dado el siguiente programa, indique cuantas veces se imprime *"Parcial"* :

```
int main(void) {
      int i;
      pid_t y = 0, x;
      for (i=0; i<2; i++) {
            x = fork();
            if (x == 0)
                  y = fork();
            if (y == 0) {
                  fork();
                  printf("Parcial.\n");
            }
      }
      exit(0);
}
```

b) Dibuje el árbol de procesos resultante de la ejecución del programa del punto 1.a.

5) Considere los siguientes programas. Si *execlp()* no falla, indique cual es la salida del programa *1c.* Si *execvp()* falla indique cual es la salida del programa *1c*.

```
/* Programa 1c /*
int main(void) {
      pid_t pid;
      pid = fork();
      if(pid == 0) {
            printf("Inicio\n");
            execlp("./prueba", "./prueba", "hola", 0);
            printf("Final\n");
      } else {
            wait(NULL);
            printf("Hijo terminado\n");
      }
      exit(0);
}

/* Programa prueba */
int main(int argc, char * argv[]) {
      printf("Valor %s", argv[1]);
      exit(0);
}
```

6) Indique cuantas veces se imprime cada valor en el siguiente programa:

```
int main (void) {
      fork();
      printf("0");
      if (fork() == 0) {
            /* el siguiente exec imprime "1" */
            execl("/bin/echo", "echo", "1", NULL);
            fork();
            printf("2");
      }
      printf("3");
      return 0;
}
```

7) Dado los siguientes programas, indique cuantas veces se imprime cada letra:

```
int main(int argc, char *argv[]) {
      int rc;
      int status;
      char *args[3];
      args[0] = (char *) "pgm";
      args[1] = (char *) "X";
      args[2] = (char *) 0;
      rc = fork();
      printf("A\n");
      if (rc != 0) {
            rc = fork();
      }
      printf("B\n");
      rc = fork();
      printf("C\n");
      if (rc == 0) {
            rc = execv("pgm", args);
            printf("D\n");
      } else {
            waitpid(rc, &status, 0);
      }
      printf("E\n");
      return 0;
}

// programa pgm:

int main(int argc, char *argv[]) {
      printf("%s\n", argv[1]);
      return 0;
}
```

8) Dado el siguiente programa indique los distintos valores que se imprimen por pantalla:

```
int tmp = 0;

void tarea (int cantidad) {
      tmp = tmp + 1;
      printf ("%d : %d\n", cantidad, tmp);
}

int main (void) {
      int f = 1, i;
      for (i = 0; i < 3; i++) {
            if (f > 0) {
                  f = fork();
            }
            if (f == 0) {
                  break;
            }
      }
      if (f == 0) {
            tarea(i);
      } else {
            printf ("main a creado %d procesos.\n", i);
      }
```

```
      return 0;
}
```

9) Cada uno de los siguientes programas imprime una declaración. Es necesario marcar *Verdadero* o *Falso* para cada programa, de acuerdo con si la declaración impresa por el programa es verdadera o falsa.

`int main(void) {`
`  if (fork() == 0) {`
`  sleep(1);`
`  printf("Seré un proceso zombie.\n");`
`  }`
`  exit(0);`
`}`
	Verdadero:
	
	Falso:

`int main(void) {`
`  int i = 5;`
 	
`  if (fork() != 0) {`
`  i += 5;`
`  sleep(2);`
`  } else {`
`  sleep(1);`
`  printf("En mi proceso, i es igual a 5.");`
`  }`
`  return 0;`
`}`
	Verdadero:
	
	Falso:

`int main(void) {`
`  int estado;`
`  if (fork() == 0) {`
`  sleep(1);`
`  printf("Seré un proceso zombie.\n");`
`  } else {`
`  wait(&estado);`
`  }`
`  exit(0);`
`}`
	Verdadero:
	
	Falso:

10) Indique cuantas veces se imprime cada valor en el siguiente programa:

```
int main (void) {
      fork();
      printf("0\n");
      if (fork() == 0) {
            fork();
            printf("2\n");
            exit(0);
      }
      fork();
      printf("4\n");
      exit(1);
      printf("3\n");
      return 0;
}
```

11) Indique cuantas veces se imprime cada valor en el siguiente programa:

```
int main(int argc, char *argv[]) {
      int pid1, pid2, pid3;

      pid1 = fork() ;
      pid2 = fork() ;
      if(pid1 == 0 ) {
            printf("A\n") ;
      } else {
            pid3 = fork() ;
            printf("B\n") ;
      }
      if (pid2 != 0 && pid3 != 0) {
            printf("C\n");
      }
      return 0;
}
```

12) Dado el siguiente programa indique cuál es la salida exacta.

```
int main (void) {
      int contador = 10;
      if (fork() == 0) {
            contador *= 2;
      } else {
            wait(NULL);
      }
      printf("%d ", ++contador);
      exit(0);
}
```

13) Indique cuantas veces se imprime 1 y 2 en el siguiente programa:

```
void realizar() {
      fork();
      fork();
      printf("1\n");
      return;
}

int main(void) {
      realizar();
      printf("2\n");
      exit(0);
}
```

14) Indique cuantas veces se imprime “Hola” en el siguiente programa:

```
int main(void) {
      int i;
      for (i = 0; i < 3; i++) {
            fork();
      }
      printf("Hola\n");
      return 0;
}
```

15) Considerando que ninguna de las llamadas al sistema o funciones fallan y las impresiones por pantalla se realizan de manera inmediata:

a) Indique cual es el árbol de procesos resultante:

```
int main(void) {
	int pid1 = fork();
	int pid2 = fork();
	if (pid2 == 0) {
		int pid3 = fork();
	} else {
		int pid5 = fork();
	}
	return 0;
}
```

b) Cuantas veces se imprimen A, B y C:

```
int main(void) {
	if (fork()== 0) {
		printf("A");
	} else {
		if (fork()) == 0) {
			fork();
			printf("B");
			fork();
			raise(SIGKILL);
			printf("C");
		}
	}
	printf("C");
	return 0;
}
```

Ejercicios a Implementar:

1) Realice un programa que cree 2 procesos hijos, donde:
- El primero debe imprimir su *PID, PPID, SID* y *PGID*, esperar 5 segundos y terminar.
- El segundo debe imprimir su *PID* y luego ejecutar el comando *ls -l*.

El proceso padre después de crear los dos procesos hijos debe finalizar.

Responda:
- ¿Existen procesos zombies? Si la respuesta es afirmativa, indique cuáles.
- ¿Existen procesos huérfanos? Si la respuesta es afirmativa, indique cuáles.

Consideraciones:
- No se puede utilizar la llamada al sistema *system()*.

2) Escriba un programa que reciba una cantidad variable de nombres de archivo desde la línea de comandos. Por cada nombre de archivo el programa debe crear un proceso hijo que:
- Imprima su *PGID*, *PPID*, *PID* y el nombre del archivo; y
- Use una de las funciones de la familia *exec()* para ejecutar: *wc -l nombre_de_archivo*.

Consideraciones:
- Los procesos hijos se deben ejecutar concurrentemente.
- El proceso padre debe indicar que proceso hijo va finalizando e indicando de que manera finaliza *(Normal o por una señal)*.
- Si no se pasa ningún nombre de archivo se debe mostrar un mensaje de error.
- No se puede utilizar la llamada al sistema *system()*.

3) Desarrolle un programa que imprima su *PID*, cree un nuevo grupo de procesos y dos procesos hijos. El primer hijo debe ejecutar el comando *whoami*. El segundo hijo debe ejecutar el comando *w*. Los procesos hijos se deben ejecutar secuencialmente. El proceso padre debe finalizar su ejecución una vez que sus procesos hijos terminan.

Consideraciones:
- No se puede utilizar la llamada al sistema *system()*.

4) Desarrolle un programa que reciba dos parámetros a través de la línea de comandos, el primero debe indicar el tipo de operación y el segundo debe ser un valor entero *(N)* que indica la cantidad de procesos a crear.

El tipo de operación indica la forma de crear los procesos: si es *"a"* el programa debe crear un hijo, el cual a su vez creará otro, y así sucesivamente, de modo que cada uno desciende del anteriormente creado, hasta llegar a crear *N* procesos; si es *"b"* el programa debe crear los *N* procesos, todos con un padre en común.

Cada hijo debe mostrar su SID, PGID, PPID, PID y terminar.

Se debe validar la cantidad de parámetros enviados al programa, si no coinciden con lo solicitado se debe mostrar un mensaje de error y terminar.

5) Realice un programa que cree dos procesos hijos (*hijo1* e *hijo2*), duerma 20 segundos y finalice. El *hijo1* debe crear un hijo (*hijo3*), transformarse en líder de un grupo de procesos, crear otro hijo (*hijo4*) y finalizar. Los procesos *hijo2*, *hijo3* e *hijo4* deben dormir 10 segundos y terminar.

Responda:
- Indique que procesos corresponden al grupo de procesos creado por el *hijo1*.
- ¿Existen procesos zombies? Si la respuesta es afirmativa, indique cuáles.
- ¿Existen procesos huérfanos? Si la respuesta es afirmativa, indique cuáles.

6) Realice un programa que genere 2 hijos, donde:
- El primero debe imprimir el *PID, PPID, UID* y *EUID*, esperar 10 segundos y terminar.
- El segundo debe ejecutar el comando *ps*.

El proceso padre debe esperar por sus dos hijos, indicando cual va terminando y luego finalizar.

Consideraciones:
- No se puede utilizar la llamada al sistema *system()*.

7) Desarrolle un programa que reciba como parámetros:
- Un programa a ejecutar y
- Un valor numérico.

El programa debe verificar que el valor numérico sea mayor que cero, de no cumplirse esta condición debe finalizar su ejecución.
Luego debe ejecutar 10 iteraciones que cumplan lo siguiente:
- Ejecutar el programa pasado como primer parámetro.
- Dormir el tiempo especificado como segundo parámetro.

8) Analice lo siguiente:
When a new process is created via fork(), the kernel ensures not only that it has a unique process ID, but also that the process ID doesn't match the process group ID or session ID of any existing process. Thus, even if the leader of a process group or a session has exited, a new process can't reuse the leader's process ID and thereby accidentally become the leader of an existing session or process group.

Que implicancias tiene dicha afirmación.

9) Lea del capítulo 6.7 del libro de Kerrrisk sobre el manejo de las variables de entorno. Responda:

a) Como debe cambiar la cabecera de main() para acceder a las variables de entorno del proceso. Que problema existe con esta forma de acceder a las varibles de entorno.

b) Estudie el uso de *getenv()*, *putenv()*, *setenv()*, *unsetenv()* y *clearenv()*.

c) Para hacer uso de estas funciones implemente un programa que después de limpiar el entorno, agrega las definiciones de entorno proporcionadas como argumentos de línea de comandos y finalmente, imprime la lista de entorno actual.Ejemplo:
$ ./modificar_env "GREET = Guten Tag" SHELL=/bin/bash BYE=Ciao

d) Implemente un programa que imprima la lista de entorno, quite del entorno las variables pasadas como argumentos de línea de comandos y vuelva a imprimir la lista de entorno. Ejemplo:
$ ./quitar_env SHELL HOME PATH

Investigación

1) En una terminal ejecute los siguientes comandos.:

a) top
b) pstree
c) pgrep init
d) ps -aux | grep root

2) En un documento denominado *capturas.odt*, por cada comando ejecutado:
- Agregue una captura de pantalla de la ejecución.
- Especifique cual considera es la función del comando.

3) Compile *anexo_1.c* y *anexo_2.c*; nombre a los binarios generados como *anexo_1* y *anexo_2* respectivamente.

En el caso del archivo *anexo_1* ejecute lo indicado en el mismo:

./anexo_1
id
sudo ./anexo_1
sudo id

En el caso del archivo *anexo_2* simplemente ejecutelo.

4) En un documento denominado *capturas_2.odt*, indique que sucede en las distintas ejecuciones tanto de *anexo_1* como *anexo_2* y agregue una captura de pantalla de cada ejecución.

5) Investigue usando las man pages si puede listar el PPID de cada proceso usando *ps*.

REFERENCIAS

Capítulo 5 de Linux System Programming, Second Edition, Robert Love, 2013.

Capítulos 6, 9, 24, 25, 26, 27 y 34 de The Linux Programming Interface, A Linux and UNIX System Programming Handbook, Michael Kerrisk. 2010