PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

1ra práctica (tipo a)
(Segundo semestre de 2011)

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) Sobre los conceptos de sistemas operativos:

a) (1 punto) Multiprogramación es una de las técnicas más importantes que se usan en sistemas operativos modernos tanto en supercomputadoras como en equipos móviles y portátiles. ¿Cómo la implementa el núcleo de un sistema operativo.

b) (1 punto) Modo *kernel* y modo de usuario. Todo lo que se ejecuta en modo *kernel* es parte del sistema operativo. ¿Y lo qué se ejecuta en modo de usuario?

Pregunta 2 (2 puntos) Sobre las llamadas al sistema:

a) (1 punto) ¿Cuál es la razón de existencia de una llamada a sistema?

b) (1 punto) Explique detalladamente cómo se implementa la función `system()` con la secuencia de llamadas al sistema `fork`, `exec`, `waitpid` y `exit`.

Pregunta 3 (2 puntos) Estructuras de sistemas operativos y máquinas virtuales:

a) (1 punto) En cuanto a la estructura de microkernel, se aboga por las ventajas de estabilidad, fiabilidad, robustez, y otras, pero, ¿a costa de qué? Señale y explique brevemente 2 desventajas inherentes de esta estructura.

b) (1 punto) El compilador Java, produce código-máquina para el procesador "Java" y no para Intel x86 o SPARC o PPC de los equipos "hardware". Entonces, ¿cómo se ejecutan dichos programas "binarios" y qué ventaja supondría compilar Java "nativamente" para alguno de los procesadores reales?

Pregunta 4 (2 puntos) Procesos y sus estados.

a) (1 punto) Un proceso "nace" cuando otro proceso "lo crea" mediante la llamada al sistema *fork()*. ¿Qué podría hacer fallar esta llamada al sistema? Mencione dos diferentes situaciones en las que *fork()* devuelva -1 (error).

b) (1 punto) ¿Qué facilita el tener un modelo de multiprogramación con PC-COUNTER (PC) o INSTRUCTION-POINTER (IP) por cada proceso? ¿En qué caso en cambio es mejor seguir la traza o hilo del PC o IP de la CPU?

Pregunta 5 (6 puntos) Se presenta el código del programa `pr1-2011-2` y los resultados de su ejecución. Se pide analizar el código proporcionado que usa las siguientes funciones y llamadas al sistema: `strncmp()` compara dos cadenas dadas analizando la cantidad de caracteres dada con el 3er argumento, y devuelve 0 en caso de coincidencia; `atoi()` convierte la cadena dada a un entero; `waitpid()` solicita la espera de terminación del proceso hijo con el PID indicado como el 1er argumento y colocación de su código de terminación en la variable indicada como el 2do argumento; la macro `WEXITSTATUS` extrae el código de terminación; `sprintf()`, en este caso, convierte el entero a una cadena de caracteres; `fork` y `execl` fueron explicadas en clase.

```
    $ cat pr1-2011-2
1   #include <unistd.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <string.h>
5   #include <sys/types.h>
```

```
 6   #include <sys/wait.h>
 7
 8   int
 9   main(int argc, char **argv)
10   {
11     int res, p1, p2, status; char str[2];
12
13     if ( (argc == 2) && (strncmp(argv[0],"test",4) == 0) ) {
14       if ( (res=atoi(argv[1]) ) == (1+2+3) ) {
15         printf("This is the full and correct mark\n");
16         exit(0);
17       }
18       printf("Something is wrong...\n");
19       exit(1);
20     }
21
22     if ( (p1=fork()) * (p2=fork()) ) waitpid(p1,&status,0);
23     if (p1 && p2) {
24       res = WEXITSTATUS(status);
25       waitpid(p2,&status,0);
26       res = res + WEXITSTATUS(status) + 1;
27       printf("My mark is %d\n", res);
28       sprintf(str, "%d", res);
29       execl("./pr1-2011-2", "test", str, (char *) NULL);
30     }
31     if (p2) waitpid(p2,&status,0);
32     p1 = p1? 2 : 0; p2 = p2? 1 : 0;
33     /* 00 = 0, 01 = 1, 10 = 2 */
34     exit(p1+p2+1);
35   }
     $ gcc pr1-2011-2.c -o pr1-2011-2
     $ ./pr1-2011-2
     My mark is 6
     This is the full and correct mark
     $
```

a) (2 puntos) Explique el mecanismo de creación de procesos en este programa y presente el crecimiento del árbol de procesos en tiempo.

b) (3 puntos) Explique el comportamiento de cada proceso (antes que se ejecuta `execl`), los valores de las variables `p1` y `p2` en cada uno de ellos y sus códigos de terminación.

c) (1 puntos) Explique la ejecución del código de programa a partir de la línea 29.

<u>Pregunta 6</u> (6 puntos) Los dos programas siguientes, sdos2unix.c y msdos2unix.c trabajan en conjunto utilizando también el programa *hexdump* , utilitario común de *NIX/GNU que produce volcado hexadecimal y los caracteres ASCII imprimibles con la opción '-*c*'

Archivo fuente: sdos2unix.c

```
#include <stdio.h>
#include <ctype.h>
#include <fcntl.h>

#define STANDARD_IN 0

#define CR '0x0D'
#define DOSEOF '0x1A'

int main(int argc, char *argv[]) {
      int c,fd;

      if (argc == 2) {
            fd = open(argv[1],O_RDONLY);
            dup2(fd,STANDARD_IN);
            close(fd);
      }

      while ((c=getchar()) != EOF)
            if ((c != CR && c != DOSEOF))
                  putchar (c);
      return 0;
}
```

Archivo fuente: msdos2unix.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv) {

      int np,i,status,fd[2];
      pid_t childpid;

      if(argc == 1) {
            fprintf(stderr,"%s: file1 [file2 ..]\n",argv[0]);
            exit(1);
      }
      pipe(fd);
      np = argc-1;

      for(i=1;i<=np;i++) {
            if((childpid=fork())==-1) {
                  fprintf(stderr,"%s: fork %d failed\n",argv[0],i);
                  exit(2);
            }
            else if (childpid==0){
                  dup2(fd[1],STDOUT_FILENO);
                  close(fd[0]);
                  close(fd[1]);
                  execl("./sdos2unix","sdos2unix",argv[í],NULL);
                  fprintf(stderr,"%s: execl failed\n",argv[0]);
                  exit(3);
            }
      }
      while(np--) wait(&status);

      if((childpid=fork() == -1) {
            fprintf(stderr,"%s: hexdump fork failed\n",argv[0]);
            exit(4);
      }
      else if (childpid==0) {
            dup2(fd[0],STDIN_FILENO);
            close(fd[0]);
            close(fd[1]);
            execl("/usr/bin/hexdump","hexdump",”-c”,NULL);
            fprintf(stderr,"%s: hexdump exec failed\n",argv[0]);
            exit(5);
      }
      close(fd[0]);close(fd[1]);
      wait(&status);
      return 0;
}
```

a) (1 punto) En base al código fuente de *sdos2unix.c* indique dos posibles formas correctas de invocar desde el *shell* el programa ya compilado *sdos2unix.* ¿Qué verificación, mensaje al usuario y resultado debería agregarse a *sdos2unix.c* para asegurar su correcto uso*?*

b) (1 punto) Explique el funcionamiento general de *sdos2unix* en base al código fuente proporcionado, sin traducir literalmente línea por línea o instrucción por instrucción.

c) (2 puntos) Haga un esquema de los procesos y tuberías creadas por *msdos2unix* si se pasan 3 nombres de archivos de texto como argumentos de su línea de comando, indicando claramente la relación padre – hijo y el flujo de datos en las tuberías mediante flechas, los descriptores de archivo fd[x] en los extremos de las tuberías y los nombres de los programas que ejecuta cada proceso.

d) (2 puntos) ¿En qué momento se espera por los hijos? Establezca una relación temporal entre los procesos creados por *msdos2unix* y considerando que las tuberías no tienen una capacidad “infinita” sino unos cuantos bloques de disco (unos 40 KiBytes), bloqueándose en espera al proceso que escriba en una tubería “llena”, señale qué problema puede ocurrir con archivos grandes, además de no aprovecharse la multitarea completamente.

------- o -------

La preguntas han sido preparadas por VK (1,2,5) y FS (3,4,6).

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

Pando, 7 de septiembre de 2011