PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
1ra práctica (tipo a)
(Segundo semestre de 2013)

Horario 0781: prof. V. Khlebnikov
Horario 0782: prof. F. Solari A.

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) (*AST – MOS3E, Chapter 1*)

a) (1 punto – 5 min.) ¿Qué características del *hardware* del computador son muy importantes para la implementación de un sistema operativo multitarea? Mencione por lo menos dos (2) y la razón de su importancia.

b) (1 punto – 5 min.) La llamada al sistema *clone()* del sistema Linux, es similar a *fork()* de otros sistemas tipo Unix. Describa brevemente como puede reducirse el uso de recursos y el tiempo de creación de un nuevo proceso con *clone()*.

c) (2 puntos – 10 min.) La función *system("línea de comando del shell")* se usa para desde un programa ejecutar la cadena "*linea de comando del shell"* precisamente como eso. El shell, que es el programa *sh*, puede ser ejecutado con argumento -c y a continuación la "*línea de comando"*. Describa entonces, como puede escribirse la función de librería *int system(char * line)*.

Pregunta 2 (6 puntos – 30 min.) Para el siguiente programa y el inicio de su ejecución indicado, presente el árbol de procesos (2 puntos), indique cómo se evaluará la condición de `if` en cada proceso y cuál será su comportamiento (2 puntos), explique cómo funcionará `while` con `waitpid()` en cada proceso (1 punto) y el papel del 2do (el último) `exit()`.

```
$ cat 2.c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>

extern char *program_invocation_name, *program_invocation_short_name;

int
main(int argc, char **argv)
{
    int n;
    char str_n[2];

    printf("%s %s\n", program_invocation_name, program_invocation_short_name);

    if ( program_invocation_short_name[1] ) exit(EXIT_FAILURE);

    n = atoi(program_invocation_short_name);
    sprintf(str_n, "%d", !n);

    if ( n && !fork() && !fork() )
        execl(program_invocation_name,str_n,NULL);
    else
        sleep(n);
        while ( waitpid(-1, NULL, 0) != -1 );
        execl("/bin/ps","ps","-l",NULL);
        exit(EXIT_FAILURE);
}

$ gcc -o 2 2.c

$ ./2
./2 2
...
```

Pregunta 3 (6 puntos – 30 min.) (*AST – OSDI3E, Minix Tests*) Los tests de Minix son pequeños programas que pretenden probar las funcionalidades de las llamadas al sistema o *System Calls*. El siguiente código es un extracto del *test2.c* de Minix. Asuma variables globales del tipo apropiado para aquellas que no son locales.

```
void test2a()
{
/* Test pipes */

  int fd[2];
  int n, i, j, q = 0;
  char buf[2048];

  subtest = 1;
  if (pipe(fd) < 0) {
        printf("pipe error.  errno= %d\n", errno);
        errct++;
        quit();
  }
  i = fork();
  if (i < 0) {
        printf("fork failed\n");
        errct++;
        quit();
  }
  if (i != 0) {
        /* Parent code */
        close(fd[0]);
        for (i = 0; i < 2048; i++) buf[i] = i & 0377;
        for (q = 0; q < 8; q++) {
                if (write(fd[1], buf, 2048) < 0) {
                        printf("write pipe err.  errno=%d\n", errno);
                        errct++;
                        quit();
                }
        }
        close(fd[1]);
        wait(&q);
        if (q != 256 * 58) {
                printf("wrong exit code %d\n", q);
                errct++;
                quit();
        }
  } else {
        /* Child code */
        close(fd[1]);
        for (q = 0; q < 32; q++) {
                n = read(fd[0], buf, 512);
                if (n != 512) {
                        printf("read yielded %d bytes, not 512\n", n);
                        errct++;
                        quit();
                }
                for (j = 0; j < n; j++)
                        if ((buf[j] & 0377) != (kk & 0377)) {
                                printf("wrong data: %d %d %d \n ",
                                                j, buf[j] & 0377, kk & 0377);
                        } else {
                                kk++;
                        }
        }
        exit(58);
  }
}
```

a) (1 punto – 5 min.) Describa en forma general y concreta lo que hace este programa, sin traducir las líneas de código.

b) (2 puntos – 10 min.) ¿Qué comprobaciones se realizan sobre la operación descrita en a) ?

El siguiente código corresponde a otro test de Minix. También asuma variables globales y que la función e() contabiliza el error, imprimiendo el valor de errno que corresponde.

```
void test40b()
{
  register int nlink;
  char bar[30];
  struct stat st, st2;

  subtest = 2;
```

```
  /* Clean up any residual */
  System("rm -rf ../DIR_40/*");

  /* Test what happens if we make LINK_MAX number of links. */
  System("touch foo");
  for (nlink = 2; nlink <= LINK_MAX; nlink++) {
        sprintf(bar, "bar.%d", nlink);
        if (link("foo", bar) != 0) e(2);
        Stat(bar, &st);
        if (st.st_nlink != nlink) e(3);
        Stat("foo", &st);
        if (st.st_nlink != nlink) e(4);
  }

  /* Check if we have LINK_MAX links that are all the same. */
  Stat("foo", &st);
  if (st.st_nlink != LINK_MAX) e(5);
  for (nlink = 2; nlink <= LINK_MAX; nlink++) {
        sprintf(bar, "bar.%d", nlink);
        Stat(bar, &st2);
        if (!stateq(&st, &st2)) e(6);
  }

  /* Test no more links are possible. */
  if (link("foo", "nono") != -1) e(7);
  if (stat("nono", &st) != -1) e(8);
  Stat("foo", &st);
  if (st.st_nlink != LINK_MAX) e(9);   /* recheck the number of links */

  /* Now unlink() the bar.### files */
  for (nlink = LINK_MAX; nlink >= 2; nlink--) {
        sprintf(bar, "bar.%d", nlink);
        Stat(bar, &st);
        if (st.st_nlink != nlink) e(10);
        Stat("foo", &st2);
        if (!stateq(&st, &st2)) e(11);
        if (unlink(bar) != 0) e(12);
  }
  Stat("foo", &st);
  if (st.st_nlink != 1) e(13); /* number of links back to 1 */

  /* Test max path ed. */
  if (link("foo", MaxName) != 0) e(14);        /* link to MaxName */
  if (unlink(MaxName) != 0) e(15);     /* and remove it */
  MaxPath[strlen(MaxPath) - 2] = '/';
  MaxPath[strlen(MaxPath) - 1] = 'a';  /* make ././.../a */
  if (link("foo", MaxPath) != 0) e(16);        /* it should be */
  if (unlink(MaxPath) != 0) e(17);     /* (un)linkable */

  System("rm -f ../DIR_40/*"); /* clean cwd */
}
```

c) (2 puntos – 10 min.) ¿Qué comprobaciones se realiza sobre LINK_MAX?

d) (1 punto – 5 min.) ¿Qué comprobación hace la función *stateq(arg1, arg2)* antes de hacer *unlink(filename)*?

<u>Pregunta 4</u> (4 puntos – 20 min.) (*AST – MOS3E, Chapter 1, 1.7 Operating System Structure*)

a) (1 punto) El núcleo de un sistema operativo puede tener estructura monolítica, o puede tener una estructura de micronúcleo entre otras. Y la última tiene unas ventajas como una alta fiabilidad, por ejemplo. Pero el núcleo es un software. Entonces, ¿se puede aplicar el enfoque de micronúcleo a cualquier software aplicativo? ¿Cuál es su opinión?

b) (1 punto) ¿Cómo se puede minimizar un núcleo usando el concepto de "mecanismo y política"? Presente el ejemplo de A. S. Tanenbaum.

c) (1 punto) A la diferencia del mecanismo de anillos de MULTICS, el esquema de capas del sistema THE se acababa con el enlazamiento de todas las partes del sistema a un solo programa ejecutable, igualmente al enfoque monolítico. Entonces, ¿cuál fue su ventaja?

d) (1 punto) ¿Cuál es la diferencia entre los monitores de máquina virtual (o *hypervisors*) del tipo 1 y del tipo 2?

La práctica ha sido preparada por FS (1,3) y VK(2,4).

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

Pando, 10 de septiembre de 2013