PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
1ra práctica (tipo a)
(Segundo semestre de 2014)

Horario 0781: prof. V. Khlebnikov

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 – 10 min.) (*MOS4E, Chapter 12*) "While layering has its supporters among system designers, another camp has precisely the opposite view. Their view is based on the **end-to-end argument**. This concept says that if something has to be done by the user program itself, it is wasteful to do it in a lower layer as well.

Consider an application of that principle to remote file access. If a system is worried about data being corrupted in transit, it should arrange for each file to be checksummed at the time it is written and the checksum stored along with the file. When a file is transferred over a network from the source disk to the destination process, the checksum is transferred, too, and also recomputed at the receiving end. If the two disagree, the file is discarded and transferred again.

This check is more accurate than using a reliable network protocol since it also catches disk errors, memory errors, software errors in the routers, and other errors besides bit transmission errors. The end-to-end argument says that using a reliable network protocol is then not necessary, since the endpoint (the receiving process) has enough information to verify the correctness of the file. The only reason for using a reliable network protocol in this view is for efficiency, that is, catching and repairing transmission errors earlier.

The end-to-end argument can be extended to almost all of the operating system. It argues for not having the operating system do anything that the user program can do itself. For example, why have a file system? Just let the user read and write a portion of the raw disk in a protected way. Of course, most users like having files, but the end-to-end argument says that the file system should be a library procedure linked with any program that needs to use files. This approach allows different programs to have different file systems. This line of reasoning says that all the operating system should do is securely allocate resources (e.g., the CPU and the disks) among the competing users."

What is the name of the operating systems structure corresponding to this view?

Pregunta 2 (7 puntos – 35 min.) En la familia de los procesos creados por el siguiente programa algunos procesos respetan la Ley de De Morgan y otros no. Explique el cálculo de la variable `r` en cada proceso (**3 puntos**), presente el árbol de procesos (**3 puntos**) e indique qué grupo de procesos acaba primero, qué grupo le sigue, etc. (**1 punto**)

```
$ cat deMorgan2.c
#include <stdlib.h>
#include <stdio.h>

#define a fork()
#define b fork()

int
main(void)
{
  int r;

  printf("Shell PID = %d\n", getppid());
  r = ( !( a || b ) == (!a && !b) );
  while ( waitpid(-1, NULL, NULL) != -1);
  printf("%d->%d: %s\n", getppid(), getpid(), r?"True":"False");
  exit(r);
}

$ gcc deMorgan2.c -o deMorgan2

$ ./deMorgan2
...
```

Pregunta 3 **(8 puntos – 40 min.)** Usted sabe como funcionan las llamadas al sistema: `pipe()` creará una tubería y dejará dos descriptores en la dirección indicada como argumento – uno para lectura y el otro para escritura; `fork()` creará un proceso hijo; `dup2()` copiará el descriptor indicado como el primer argumento al descriptor indicado como el segundo argumento; `exec()` reemplazará el programa del proceso por el programa cuyo nombre y la ruta se indican como argumentos; `waitpid(-1,...)` pondrá el proceso en espera de la terminación de cualquier hijo y devolverá el pid del hijo cuando éste termine, o devolverá -1 si ya no hay ningún hijo. Ahora explique qué árbol de procesos se formará durante la ejecución de este programa (**3 puntos**), cuál será la comunicación entre los procesos (**2 puntos**), y qué hará cada proceso durante su ejecución (**3 puntos**):

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(void);
void die(char *s);

int main(void)
{
  int pfd[2], n=2, i=0;
  pid_t pid, ppid;

  ppid=getppid(); /* ¿cuál es el pid de mi padre? */

  /* o tendré mi tubería, o me muero */
  if (pipe(pfd) == -1) die("pipe() error\n");

  /* me moriré si no tendré un hijo */
  if ((pid=fork()) == -1) die("fork() error\n");
  if (!pid) {
    if (dup2(pfd[1],1) == -1) die("dup2(pfd[1],1) error\n");
    (void) close(pfd[0]); /* el error de close() no nos interesa */
    sleep(1);
    execl("/bin/ps", "ps", "-l", NULL);
    die("execl(/bin/ps) error\n");
  }

  while(i<n && (fork() || !fork())) i++;
  sleep(3);
  while(waitpid(-1, NULL, 0) != -1);

  if (ppid != getppid()) exit(0);

  if (dup2(pfd[0],0) == -1) die("dup2(pfd[0],0) error\n");
  (void) close(pfd[1]);
  execl("/bin/more", "more", NULL);
  die("execl(/bin/more) error\n");
}


void die(char *s)
{
  if (s != (char *)NULL) {
    while (*s) (void) write(2, s++, 1);
  }
  exit((s == (char *) NULL) ? 0 : 1);
}
```

Pregunta 4 **(3 puntos – 15 min.)** Hay 2 procesos concurrentes (o hilos) que trabajan con la misma variable entera compartida `x` que tiene su valor inicial 0. Un proceso la incrementa 2 veces en 1, el otro la decrementa 2 veces en 1. Los procesos se ejecutan al mismo tiempo y el orden de ejecución de sus instrucciones (que se intercalan) es desconocido. ¿Cuáles serán los valores finales posibles de la variable?

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 17.

Profesor del curso: (0781) V. Khlebnikov

Pando, 16 de septiembre de 2014