PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

1ra práctica (tipo a)
(Primer semestre de 2023)

Horarios 0781, 0782: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **Sin apuntes de clase y sin calculadora o computadora.**
La presentación, la ortografía y la gramática influirán en la calificación.
Puntaje total: 20 puntos

Pregunta 1 (2 puntos – 10 min.)

(a) (0,5 puntos) Para un proceso suspendido, ¿qué es su *core image*?

(b) (0,5 puntos) Para un sistema operativo, ¿en qué consiste un proceso suspendido?

(c) (1 punto) ¿Cuál es la memoria más rápida que usa un proceso y dónde se guarda su contenido cuando el proceso se suspende?

Pregunta 2 (4 puntos – 20 min.)

(a) (1 punto) Uno de los métodos usados en los sistemas operativos se llama *busy waiting*, o sea, la espera ocupada. Pero, ¿por qué la espera es ocupada? ¿Ocupada de qué?

(b) (2 puntos) Se distinguen *hard real-time systems* y *soft real-time systems*. ¿Cuál es la diferencia entre ellos? Los teléfonos celulares inteligentes, ¿pertenecen a uno de ellos o no?

(c) (1 punto) Cuando estamos hablando sobre los espacios de direcciones de los procesos, ¿estamos hablando de la memoria RAM de la computadora?

Pregunta 3 (2 puntos – 10 min.) ¿Cuáles son todos resultados posibles de la situación ilustrada en la figura 2.21? ¿Cuál es el recurso compartido?

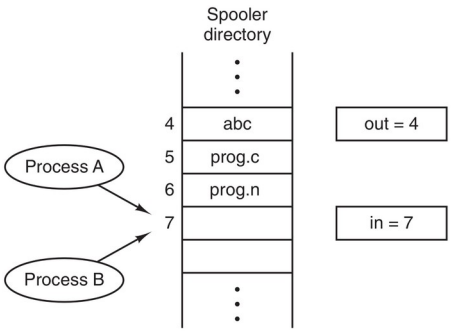

Figure 2-21. Two processes want to access shared memory at the same time.

Pregunta 4 (2 puntos – 10 min.) Se pretende la organización del trabajo intercalado de 2 hilos creados con los identificadores `th_flip` y `th_flop`, ejecutando las funciones respectivas `thread_flip()` y `thread_flop()`. Para sincronización se usan las funciones `thr_continue()` y `thr_suspend()`:

`thr_suspend()` immediately suspends the execution of the thread specified by target_thread.
On successful return from `thr_suspend()`, the suspended thread is no longer executing.
Once a thread is suspended, subsequent calls to `thr_suspend()` have no effect.
Signals cannot awaken the suspended thread; they remain pending until the thread resumes execution.

`thr_continue()` resumes the execution of a suspended thread.
Once a suspended thread is continued, subsequent calls to `thr_continue()` have no effect.

Al terminar su parte de trabajo, cada hilo despierta al otro y suspende a sí mismo:

```
...
thread_t th_flip, th_flop;
...
```

```
void * thread_flip(void *arg) {
    ...
    while (1) {
        ...
        thr_continue(th_flop);
        thr_suspend(th_flip);
    }
}
```

```
void * thread_flop(void *arg) {
    ...
    while (1) {
        thr_suspend(th_flop);
        ...
        thr_continue(th_flip);
    }
}
```

Encuentre *race condition* en este código.

Pregunta 5 (2 puntos – 10 min.) Consider the following program:

```
 A: {
      shared int x;

1     x = 10;
2     while (1) {
3         x = x - 1;
4         x = x + 1;
5         if (x != 10)
6             printf("x is %d", x);
      }
    }
```

```
 B: {
      shared int x;

1     x = 10;
2     while (1) {
3         x = x - 1;
4         x = x + 1;
5         if (x != 10)
6             printf("x is %d", x);
      }
    }
```

Note that the scheduler in a uniprocessor system would implement pseudo-parallel execution of these two concurrent processes by interleaving their instructions, without restriction on the order of the interleaving. Show a sequence (i.e., trace the sequence of interleaving of statements) such that the statement "`x is 10`" is printed.

Pregunta 6 (4 puntos – 20 min.) Considere el siguiente programa y presente el resultado de su ejecución usando los números arbitrarios pero correlativos de los PIDs de procesos:

```
$ cat process_chain_v2.c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int
main(int argc,char **argv) {
    int n, i;
    pid_t *pids;
    pid_t ppid = getppid();

    if (argc == 1) {
        printf("Falta indicar la cantidad de procesos\n");
        exit(EXIT_FAILURE);
    }
    n= atoi(argv[1]);
    if (n <= 0) exit(EXIT_SUCCESS);

    pids = (pid_t *) calloc(n,sizeof(pid_t));
    *(pids+n-1) = getpid();

    for (i=1; i<n; ++i)
        if (fork()) break;
        else *(pids+n-i-1) = getpid();

    waitpid(-1,NULL,0);
    for (i=0; i<n; i++) printf("%d\n",*(pids+i));
    printf("\n");
    exit(EXIT_SUCCESS);
}

$ gcc process_chain_v2.c -o process_chain_v2
$ ./process_chain_v2 3
...
```

Pregunta 7 (4 puntos – 20 min.) Usted puede completar el siguiente código y presentar la salida de su ejecución.

```
$ cat foo.c
#include <sys/types.h>
#include <sys/wait.h>
```

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

int
main(int argc,char **argv) {
    int n, bar, i;
    char barstr[3];

    if (argc == 1) {
        printf("Mensaje de error\n");
        exit(EXIT_FAILURE);
    }
    n= atoi(argv[1]);
    if (n <= 1) exit(EXIT_SUCCESS);
    if (argc == 2) bar= 0;
    else bar= atoi(argv[2]);
    if (bar < n) {
        sprintf(barstr,"%d", ...);
        for (i=0; i<n; ++i)
            if (...) execl("./foo", "foo", argv[1], barstr, (char *)0);
    }
    for (i=0; i<n; ++i) waitpid(-1,NULL,0);
    printf("pid=%d, ppid=%d, argv[2]=%s\n",getpid(),getppid(),argv[2]);
    exit(EXIT_SUCCESS);
}
$ gcc foo.c -o foo
$ ./foo 2
...
```

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 21.1 “Vera”

Profesor del curso: V. Khlebnikov

Pando, 20 de abril de 2023