# Sistemas Operativos – edición OSTEP

## Virtualización

**Ejercicio 1.** Explique detalladamente como funcionan estos programas indicando cuantos procesos se crean, que hacen los hijos y padres, que archivos se cierran y abren, etc.
Suponga que en ambos casos se invocan al programa compilado `./a.out` de la siguiente forma:

```
$ ./a.out A B C D E F
```

```c
int main(int argc, char ** argv) {
  int rc = fork();
  if (rc<0)
    return -1;
  else if (0==rc)
    execvp(argv[0], argv);
  else {
    execvp(argv[0], argv);
  }
}
```

```c
int main(int argc, char ** argv) {
  char buf[L] = {'\0'};
  if (argc<=1)
    return 0;
  int rc = fork();
  if (rc<0)
    return -1;
  if (0==rc) {
    close(0);
    open(argv[0], 0);
    read(0, buf, L);
    write(1, buf, L);
  } else {
    argv[argc-1] = NULL;
    execvp(argv[0], argv);
  }
}
```

**Ejercicio 2.** Para el sistema de paginado i386 $(10, 10, 12)$, sabiendo que `CR3=0x00000` y que el contenido de los marcos físicos son los siguientes:

```
0x00000                 0x10000
------------------      ------------------
0x000: 0x10000 (P)      0x000: 0x00000 (P)
0x001: 0x10000 (P)      0x001: 0x00001 (P)
...                     ...
...                     ...
0x3FE: 0x10000 (P)      0x3FE: 0x003FE (P)
0x3FF: 0x00000 (P)      0x3FF: 0x003FF (P)
```

(a) Pasar de virtual a física: `0x00000EEF`, `0x00001FEE`.

(b) Pasar de **física** a virtual: `0x00000EEF`, `0x00001FEE`.

(c) Indique qué contiene la dirección de memoria virtual `0xFFC00CFF`.

(d) Modificar el mapa de memoria virtual de forma tal que `0x00000EEF` apunte a `0x00000EEF` y `0x00400EEF` apunte a `0x00001EEF`.

Ejercicio 3. Para el programa de la Figura 1, donde la **atomicidad es línea-a-línea**:

(a) Muestre un (1) escenario de ejecución con $N = 5$ de forma tal que la salida del multiprograma cumpla con $a = [0, 0, 0, 0, 1]$.

(b) Muestre un (1) escenario de ejecución con $N = 4$ de forma tal que la salida del multiprograma cumpla con $a = [0, 0, 2, 2]$.

(c) Exprese **todos los valores posibles** de salida de arreglo `a` para la Figura 2. Explique.

(d) Sincronice el multiprograma de la Figura 1 **usando semáforos** para que el resultado **siempre** sea $a = [0, 0, ..., 0, 1]$ para cualquier valor de $0 < N$. Puede colocar condicionales (`if`) sobre el valor de `i,j` para hacer wait/post de los semáforos. Ninguna variable del programa se puede modificar.

| Pre: $0<$`N` $\wedge$ `i,j`$=0$ $\wedge$ $(\forall k : 0 \leq k < N :$ `a`$[k] = 2)$ | |
|---|---|
| `1 P0: while (i<N) {`<br>`2  a[i] = 0;`<br>`3  ++i;`<br>`}` | `a P1: while (j<N) {`<br>`b  a[j] = 1;`<br>`c  ++j;`<br>`}` |
| a=? | |

Figura 1: Concurrent Vector Writing (CVW)

| Pre: $0<$`N` $\wedge$ `i,j`$=0$ $\wedge$ `s,t`$=0, 2$ $\wedge$ $(\forall k : 0 \leq k < N :$ `a`$[k] = 2)$ | |
|---|---|
| `P0: while (i<N) {`<br>`sem_wait(s);`<br>`a[i] = 0;`<br>`++i;`<br>`sem_post(t);`<br>`}` | `P1: while (j<N) {`<br>`sem_wait(t);`<br>`a[j] = 1;`<br>`++j;`<br>`sem_post(s);`<br>`}` |
| a=? | |

Figura 2: CVW sincronizado

Ejercicio 4. Considere los procesos `P0` y `P1` a continuación, donde las sentencias son atómicas.

| Pre: `n`$= 0$ | |
|---|---|
| `P0 : while (n<100) {`<br>`n = n+1;`<br>`}` | `P1 : while (n<100) {`<br>`n = n*3;`<br>`}` |
| Post: `n`$= 66$ | |

(a) ¿Se cumple la postcondición? Demostración rigurosa o contraejemplo.

(b) Sincronizar con semáforos para que siempre dé como resultado `n=666`. Puede colocar condicionales (`if`) sobre el valor de `n` para hacer wait/post de los semáforos.

(c) Sincronizar con semáforos para que siempre dé como resultado `n=243`. Puede colocar condicionales (`if`) sobre el valor de `n` para hacer wait/post de los semáforos.

Ejercicio 5. El disco *Pepito Digital Green* de 2 TiB e intefaz SATA-3 tiene una velocidad de rotacional de 4500 RPM, 6.5 ms de latencia de búsqueda y 80 MiB/s de tasa de transferencia máxima (un disco rotacional para el olvido).

(a) Indicar cuantos *ms* tarda en dar una vuelta completa.

(b) Indicar la tasa de transferencia de lectura **al azar** de bloques de 8 MiB.

(c) Si la tasa de transferencia máxima está dada por la velocidad rotacional que no requiere cambio de pista (no sufre del *seek time*), deducir cuantos MiB almacena cada pista.

Ejercicio 6. En un sistema de archivos de tipo UNIX, tenemos los bloques de disco dispuestos dentro del *i-nodo* con 12 bloques directos, 1 bloque indirecto, 1 bloque doble indirecto y 1 bloque triple indirecto. Cada bloque es de 4 KiB y los números de bloque ocupan 32 bits.
La partición está formateada con 262144 i-nodos y una región de datos de 512 MiB.
(a) Calcule el tamaño en KiB del *inode-bitmap* y del *data-bitmap*.
(b) Suponiendo un tamaño de inodo de 128 bytes, calcule el tamaño de la tabla de inodos.
(c) ¿Es posible agotar la capacidad de almacenamiento de la partición sin almacenar un solo byte de datos?
(d) ¿A partir de que tamaño en KiB el archivo empieza a utilizar los bloques doble indirectos?