Sistemas Operativos

Práctica 1: Procesos y API del SO

Notas preliminares

- Los ejercicios marcados con el símbolo ★ constituyen un subconjunto mínimo de ejercitación. Sin embargo, aconsejamos fuertemente hacer todos los ejercicios.

Parte 1 – Estado y operaciones sobre procesos

Ejercicio 1

¿Cuáles son los pasos que deben llevarse a cabo para realizar un cambio de contexto?

Ejercicio 2 ★

El PCB (Process Control Block) de un sistema operativo para una arquitectura de 16 bits es

```
struct PCB {
  int STAT;     // valores posibles KE_RUNNING, KE_READY, KE_BLOCKED, KE_NEW
  int P_ID;     // process ID
  int PC;       // valor del PC del proceso al ser desalojado
  int R0;       // valor del registro R0 al ser desalojado
  ...
  int R15;      // valor del registro R15 al ser desalojado
  int CPU_TIME  // tiempo de ejecución del proceso
}
```

a) Implementar la rutina `Ke_context_switch(PCB* pcb_0, PCB* pcb_1)`, encargada de realizar el cambio de contexto entre dos procesos (cuyos programas ya han sido cargados en memoria) debido a que el primero ha consumido su *quantum*. `pcb_0` es el puntero al PCB del proceso a ser desalojado y `pcb_1` al PCB del proceso a ser ejecutado a continuación. Para implementarla se cuenta con un lenguaje que posee acceso a los registros del procesador R0, R1, ..., R15, y las siguientes operaciones:

```
·=·;                                // asignación entre registros y memoria
int  ke_currrent_user_time();       // devuelve el valor del cronómetro
void ke_reset_current_user_time();  // resetea el cronómetro
void ret();                         // desapila el tope de la pila y reemplaza el PC
```

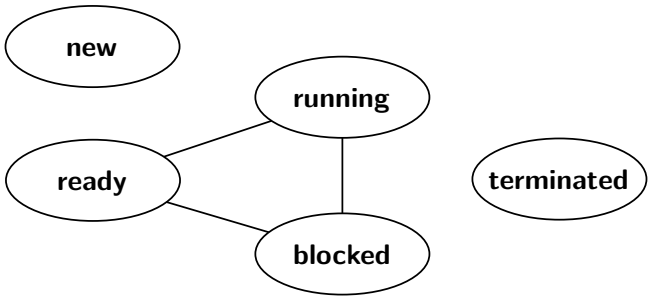
b) Identificar en el programa escrito en el punto anterior cuáles son los pasos del ejercicio 1.

Ejercicio 3

Describir la diferencia entre un *system call* y una llamada a función de biblioteca.

Ejercicio 4 ★

En el esquema de transición de estados que se incluye a continuación:



a) Dibujar las puntas de flechas que correspondan. Agregar las transiciones que crea necesarias entre los estados disconexos y el resto.
b) Explicar cuál es la causa de cada transición y qué componentes (*scheduler*, proceso, etc.) estarían involucrados.

Ejercicio 5 ★

Un sistema operativo ofrece las siguientes llamadas al sistema:

`pid fork()`	Crea un proceso exactamente igual al actual y devuelve el nuevo *process* ID en el proceso padre y 0 en el proceso hijo.
`void wait_for_child(pid child)`	Espera hasta que el *child* indicado finalice su ejecución.
`void exit(int exit_code)`	Indica al sistema operativo que el proceso actual ha finalizado su ejecución.
`void printf(const char *str)`	Escribe un *string* en pantalla.

a) Utilizando únicamente la llamada al sistema `fork()`, escribir un programa tal que construya un árbol de procesos que represente la siguiente genealogía: Abraham es padre de Homer, Homer es padre de Bart, Homer es padre de Lisa, Homer es padre de Maggie. Cada proceso debe imprimir por pantalla el nombre de la persona que representa.

b) Modificar el programa anterior para que cumpla con las siguientes condiciones: 1) Homer termine sólo después que terminen Bart, Lisa y Maggie, y 2) Abraham termine sólo después que termine Homer.

Ejercicio 6

El sistema operativo del punto anterior es extendido con la llamada al sistema `void exec(const char *arg)`. Esta llamada al sistema reemplaza el programa actual por el código localizado en el *string*. Implementar la llamada al sistema `void system(const char *arg)` usando las llamadas al sistema ofrecidas por este sistema operativo.

Parte 2 – Comunicación entre procesos

Ejercicio 7 ★

Un maniático del conocido juguete de malabares *tiki-taka*[1] ha decidido homenajear a dicho juego mediante la ejecución del siguiente programa:

- Un proceso lee la variable `tiki` y escribe su contenido incrementado en 1 en la variable `taka`.

[1] http://es.wikipedia.org/wiki/Tiki_taka

- Otro proceso lee la variable `taka`, escribiendo su contenido incrementado en 1 en la variable `tiki`.

```
#define SIZE 5

int tiki;
int taka;
int temp;

void taka_runner() {
  while (true) {
     temp = tiki;
     temp++;
     taka = temp;
  }
}

void tiki_taka() {
  while (true) {
     temp = taka;
     temp++;
     tiki = temp;
  }
}
```

El sistema operativo ofrece las siguientes llamadas al sistema para efectuar una comunicación entre distintos procesos:

`void share_mem(int *ptr)` `void share_mem(int *ptr, int size)`	Permite compartir el puntero a todos los hijos que el proceso cree.

a) ¿Qué variables deben residir en el área de memoria compartida?

b) ¿Existe alguna variable que no deba residir en el espacio de memoria compartida?

c) Escribir un procedimiento `main()` para el problema del tiki-taka usando el código presentado y las llamadas al sistema para comunicación entre procesos provistas por este sistema operativo.

Ejercicio 8 ★

Un nuevo sistema operativo ofrece las siguientes llamadas al sistema para efectuar comunicación entre procesos:

`void bsend(pid dst, int msg)`	Envía el valor `msg` al proceso `dst`.
`int breceive(pid src)`	Recibe un mensaje del proceso `src`.

Ambas llamadas al sistema son bloqueantes y la cola temporal de mensajes es de capacidad *cero*. A su vez, este sistema operativo provee la llamada al sistema `pid get_current_pid()` que devuelve el process id del proceso que la invoca.

a) Escribir un programa que cree un segundo proceso, para luego efectuar la siguiente secuencia de mensajes entre ambos:

1. *Padre* envía a *Hijo* el valor 0,

2. *Hijo* envía a *Padre* el valor 1,
3. *Padre* envía a *Hijo* el valor 2,
4. *Hijo* envía a *Padre* el valor 3,

   etc...

b) Modificar el programa anterior para que cumpla con las siguientes condiciones: 1) se cree un tercer proceso, y 2) se respete esta nueva secuencia de mensajes entre los tres procesos.

1. *Padre* envía a *Hijo_1* el valor 0,
2. *Hijo_1* envía a *Hijo_2* el valor 1,
3. *Hijo_2* envía a *Padre* el valor 2,
4. *Padre* envía a *Hijo_1* el valor 3,

   etc...

c) ¿En el punto anterior usó más mensajes que los indicados (es decir, un mensaje para el valor 1, otro para el valor 2, etc)? Si es así, reescribir el programa compartiendo esa información mediante las llamadas al sistema provistas en el ejercicio 7 para manejar memoria compartida.

Ejercicio 9

El siguiente programa se ejecuta sobre dos procesos: uno destinado a ejecutar el procedimiento `cómputo_muy_difícil_1()` y el otro destinado a ejecutar el procedimiento `cómputo_muy_difícil_2()`. Como su nombre lo indica, ambos procedimientos son sumamente *costosos* y duran prácticamente lo mismo. Ambos procesos se conocen mutuamente a través de las variables `pid_derecha` y `pid_izquierda`.

```
int result;

void proceso_izquierda() {
  result = 0;
  while (true) {
    bsend(pid_derecha, result);
    result = cómputo_muy_difícil_1();
  }
}

void proceso_derecha() {
  while(true) {
    result = cómputo_muy_difícil_2();
    int left_result = breceive(pid_izquierda);
    printf("%s %s", left_result, result);
  }
}
```

El hardware donde se ejecuta este programa cuenta con varios procesadores. Al menos dos de ellos están dedicados a los dos procesos que ejecutan este programa. El sistema operativo tiene una cola de mensajes de capacidad cero. Las funciones `bsend()` y `breceive()` son las mismas descriptas en el ejercicio anterior (ambas bloqueantes).

a) Sea la siguiente secuencia de uso de los procesadores para ejecutar los procedimientos costosos.

Tiempo	Procesador 1	Procesador 2
1	`cómputo_muy_difícil_1`	`cómputo_muy_difícil_2`
2	`cómputo_muy_difícil_1`	`cómputo_muy_difícil_2`
3	`cómputo_muy_difícil_1`	`cómputo_muy_difícil_2`
...	...	...

Explicar por qué esta secuencia no es realizable en el sistema operativo descripto. Escribir una secuencia que sí lo sea.

b) ¿Qué cambios podría hacer *al sistema operativo* de modo de lograr la secuencia descripta en el punto anterior?

Ejercicio 10

Mencionar y justificar qué tipo de sistema de comunicación (basado en memoria compartida o en pasaje de mensajes) sería mejor usar en cada uno de los siguientes escenarios:

a) Los procesos `cortarBordes` y `eliminarOjosRojos` necesitan modificar un cierto archivo `foto.jpg` al mismo tiempo.
b) El proceso `cortarBordes` se ejecuta primero y luego de alguna forma le avisa al proceso `eliminarOjosRojos` para que realice su parte.
c) El proceso `cortarBordes` se ejecuta en una casa de fotos. El proceso `eliminarOjosRojos` es mantenido en tan estricto secreto que la computadora que lo ejecuta se encuentra en la bóveda de un banco.

Ejercicio 11 ★

Un sistema operativo provee las siguientes llamadas al sistema para efectuar comunicación entre procesos mediante pasaje de mensajes.

`bool send(pid dst, int *msg)`	Envía al proceso `dst` el valor del puntero. Retorna `false` si la cola de mensajes estaba llena.
`bool receive(pid src, int *msg)`	Recibe del proceso `src` el valor del puntero. Retorna `false` si la cola de mensajes estaba vacía.

a) Modificar el programa del ejercicio 9 para que utilice estas llamadas al sistema.

b) ¿Qué capacidad debe tener la cola de mensajes para garantizar el mismo comportamiento?

Ejercicio 12

¿Qué sucedería si un sistema operativo implementara *pipes* como único sistema de comunicación interprocesos? ¿Qué ventajas tendría incorporar memoria compartida? ¿Y *sockets*?

Ejercicio 13

Pensar un escenario donde tenga sentido que dos procesos (o aplicaciones) tengan entre sí un canal de comunicaciones bloqueante y otro no bloqueante. Describir en pseudocódigo el comportamiento de esos procesos.

Ejercicio 14

El comportamiento esperado del siguiente programa es que el proceso hijo envíe el mensaje *"hola"* al proceso padre, para que luego el proceso padre responda *"chau"*. Encontrar un defecto en esta implementación y solucionarlo.

```
pid shared_parent_pid;
pid shared_child_pid;
mem_share(&shared_parent_pid);
mem_share(&shared_child_pid);
shared_parent_pid = get_current_pid();
pid child = fork();

if (child == 0) {
  shared_child_pid = get_current_pid();
  bsend(shared_parent_pid, "hola");
  breceive(shared_parent_pid);
  exit(OK);
} else {
  breceive(shared_child_pid);
  bsend(shared_child_pid, "chau");
  exit(OK);
}
```

Ejercicio 15 ★

Escribir el código de un programa que se comporte de la misma manera que la ejecución del comando "`ls -al | wc -l`" en una *shell*. No está permitido utilizar la función `system`, y cada uno de los programas involucrados en la ejecución del comando deberá ejecutarse como un subproceso.

Ejercicio 16

Se desea hacer un programa que corra sobre una arquitectura con 8 núcleos y calcule promedios por fila de un archivo de entrada que contiene una matriz de enteros positivos de $N \times M$. Se quiere que el promedio de cada fila sea calculado en un proceso separado, con un máximo de 8 procesos simultáneos, y que los procesos se comuniquen utilizando *pipes*. Cada proceso debe recibir una fila para calcular del proceso padre, quien las distribuirá entre sus hijos siguiendo una política *round-robin*. Finalmente, la salida del programa debe mostrarse por la salida estándar, ordenada de menor a mayor. Por ejemplo, si el programa recibe como entrada un archivo con la siguiente matriz (con $N = 3$ y $M = 4$):

$$\begin{pmatrix} 4 & 4 & 2 & 2 \\ 1 & 2 & 8 & 9 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

la salida debe ser $(1\ 3\ 5)^\top$.

Escribir la implementación del programa. Se puede asumir que N, M, el nombre del archivo y la cantidad de núcleos se encuentran *hardcodeados*, y que se cuenta con las siguientes funciones auxiliares:

- `int cargar_fila(const int fd, int* lista)`: lee una línea del archivo indicado por el *file descriptor* `fd` como una lista de enteros, y la almacena en `lista`. Devuelve 1 en caso de éxito, 0 si no quedan más filas.
- `int calcular_promedio(const int *lista)`: toma la lista de enteros indicada por `lista` y devuelve su promedio utilizando división entera.

- `void sort(char *s)`: toma la cadena de texto indicada por `s` y conformada por un número por línea, y la modifica de forma que quede ordenada de menor a mayor (similar a ejecutar `sort -n` en UNIX).
- `int dup2(int oldfd, int newfd)`: *linkea* los *file descriptors* `newfd` y `oldfd`, de forma tal que realizar una operación sobre `newfd` es equivalente a hacerla sobre `oldfd`.

Ejercicio 17

Se tiene un programa que cada vez que se lo ejecuta (sin parámetros) imprime lo siguiente a la salida estándar:

```
¿Cuál es el significado de la vida?
Dejame pensarlo...
Ya sé el significado de la vida.
Mirá vos. El significado de la vida es 42.
```

```
¡Bang Bang, estás liquidado!
Me voy a mirar crecer las flores desde abajo.
Te voy a buscar en la oscuridad.
```

y al correrlo con `strace` se obtiene la siguiente salida (se omiten las partes irrelevantes):

```
execve("./estrella", ["./estrella"], [/* 33 vars */]) = 0
pipe([3, 4])                            = 0
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|
     CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x15acb50) = 6590
[6590] close(3)                         = 0
[6590] getppid( <unfinished ...>
[6589] close(4)                         = 0
[6589] rt_sigaction(SIGINT, {0x40105e, [INT], ...},  <
     unfinished ...>
[6590] <... getppid resumed> )          = 6589
[6589] <... rt_sigaction resumed> {SIG_DFL, [], 0}, 8) = 0
[6590] rt_sigaction(SIGINT, {0x4010ea, [INT], ...},  <
     unfinished ...>
[6589] rt_sigprocmask(SIG_BLOCK, [CHLD],  <unfinished ...>
[6590] <... rt_sigaction resumed> {SIG_DFL, [], 0}, 8) = 0
[6589] <... rt_sigprocmask resumed> [], 8) = 0
[6590] rt_sigaction(SIGHUP, {0x40115d, [HUP], ...},  <
     unfinished ...>
[6589] rt_sigaction(SIGCHLD, NULL,  <unfinished ...>
[6590] <... rt_sigaction resumed> {SIG_DFL, [], 0}, 8) = 0
[6589] <... rt_sigaction resumed> {SIG_DFL, [], 0}, 8) = 0
[6589] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
[6589] nanosleep({1, 0}, 0x7ffdd87913d0) = 0
[6589] fstat(1, ...}) = 0
[6589] mmap(NULL, 4096, PROT_READ|PROT_WRITE, ...) = 0x7f4
[6589] write(1, "¿Cuál es el significado de la "..., 38) = 38
[6589] kill(6590, SIGINT <unfinished ...>
[6590] --- SIGINT {si_signo=SIGINT, si_code=SI_USER, si_pid
     =6589}
[6589] <... kill resumed> )             = 0
[6590] fstat(1, ...}) = 0
[6590] mmap(NULL, 4096, PROT_READ|PROT_WRITE, ...) = 0x7f4
[6590] write(1, "Dejame pensarlo...\n", 19) = 19
[6590] rt_sigprocmask(SIG_BLOCK, [CHLD], [INT], 8) = 0
[6590] rt_sigaction(SIGCHLD, NULL, {SIG_DFL, [], 0}, 8) = 0
[6590] rt_sigprocmask(SIG_SETMASK, [INT], NULL, 8) = 0
```

```
[6590] nanosleep({5, 0}, 0x7ffdd8790e00) = 0
[6590] write(1, "Ya sé el significado de la vida"..., 34) = 34
[6590] write(4, "42", 2)                = 2
[6590] kill(6589, SIGINT)               = 0
[6589] --- SIGINT {si_signo=SIGINT, si_code=SI_USER, si_pid
     =6590} ---
[6590] rt_sigreturn()                   = 0
[6589] read(3, "42", 3)                 = 2
[6589] write(1, "Mirá vos. El significado de la "..., 44) = 44
[6589] write(1, "¡Bang Bang, estás liquidado!\n", 31) = 31
[6589] kill(6590, SIGHUP <unfinished ...>
[6590] --- SIGHUP {si_signo=SIGHUP, si_code=SI_USER, si_pid
     =6589} ---
[6589] <... kill resumed> )             = 0
[6589] rt_sigprocmask(SIG_BLOCK, [CHLD], [INT], 8) = 0
[6590] write(1, "Me voy a mirar crecer las flores"..., 46 <
     unfinished ...>
[6589] rt_sigaction(SIGCHLD, NULL, {SIG_DFL, [], 0}, 8) = 0
[6589] rt_sigprocmask(SIG_SETMASK, [INT],  <unfinished ...>
[6590] <... write resumed> )            = 46
[6589] <... rt_sigprocmask resumed> NULL, 8) = 0
[6590] close(4)                         = 0
[6590] exit_group(0)                    = ?
[6589] nanosleep({10, 0},  <unfinished ...>
[6590] +++ exited with 0 +++
<... nanosleep resumed> {10, 32866})    = ?
     ERESTART_RESTARTBLOCK (Interrupted by signal)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=6590,
      si_status=0, si_utime=100, si_stime=0} ---
restart_syscall(<... resuming interrupted call ...>) = 0
write(1, "Te voy a buscar en la oscuridad.\n", 33) = 33
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

a) Identificar qué funciones de la `libc` generan cada una de las *syscalls* observadas.
b) Escribir un programa que posea un comportamiento similar al observado. Es decir que, al ejecutarlo, produzca la misma salida, y que la secuencia de *syscalls* observadas al correrlo con `strace` sea la misma que se muestra aquí.