



Sistemas Operativos y Distribuidos



Anexo práctico 2 Casos de estudio



GNU/Linux

La realización del anexo GNU/Linux es obligatorio. Es importante la comprensión de estos contenidos dado que serán necesarios para la resolución de los próximos anexos.

A.L.2.1. Comando `ps`.

- Ejecutar el comando `ps` (acrónimo de "*process status*") sin opciones, ¿qué procesos muestra?
- Ejecutar el comando de tal manera que muestre todos los procesos.
- Ejecutar el comando de tal manera que muestre el árbol de procesos. Contrastar esta información con la que muestra el comando `pstree`.
- Ejecutar el comando de tal manera que muestre todos los procesos, nombre de usuario, *ID* de usuario, comando, *PID* y estado del proceso.

A.L.2.2. Ejecutar el comando `sleep 120` y suspenderlo con Ctrl-z. Luego ejecutar el comando `ping localhost` y suspenderlo con Ctrl-z. Utilizando el comando `fg`, traer a primer plano el segundo comando suspendido y finalizarlo. A continuación, traer a primer plano el primer comando ejecutado y esperar su finalización.

A.L.2.3. Ejecutar el comando `sleep 60` en segundo plano con la opción `&`. Luego utilizar el comando `kill` para finalizarlo por su número de *job*. Repetir el proceso, pero en esta oportunidad finalizarlo por su *PID*.

A.L.2.4. El comando `top` (acrónimo de *table of processes*) se encuentra en la mayoría de los sistemas operativos de tipo Unix y muestra una lista ordenada de los procesos que se están ejecutando. Sin embargo, actualmente tiene un reemplazo más interactivo, el comando `htop`. Ejecutar dicho comando para identificar la siguiente información:

- En la barra representando la carga de *cpu/core*; explicar cuál es el significado de los colores de la misma.
- En la barra representando la memoria; explicar cuál es el significado de los colores de la misma.
- Ejecutar el comando `sleep 500` en segundo plano y luego finalizarlo utilizando `htop`.

A.L.2.5. Los “Pipes” se utilizan muy a menudo en el intérprete de comandos, para situaciones en las cuales la salida de un comando sirve como entrada para otro. Un “Pipe” se puede construir en la línea de comandos con el carácter “|”. A continuación, crear un “pipeline” paso a paso agregando sólo un comando por vez y explicando en cada inciso cuál es la utilidad del comando agregado:

- a) `ps -fe`
- b) `ps -fe | more`
- c) `ps -fe | grep root | more`
- d) `ps -fe | grep root | sort | more`
- e) `ps -fe | grep root | sort | pr -h "Procesos de Root" | more`

A.L.2.6. ¿Cuál es la utilidad del comando: `ps -ef | awk '{print $8}'` ?

A.L.2.7. Investigar qué son los procesos *Zombie* y *Orphan*.

A.L.2.8. Ejecutar el siguiente programa y responder cuántas veces imprime “!Hello World!”?

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

int main(void) {
    fork();
    fork();
    fork();
    puts("!Hello World!");
    return EXIT_SUCCESS;
}
```

A.L.2.9. ¿Cuántos procesos son creados al ejecutar el siguiente código? Dibujar el árbol de procesos.

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

int main(void) {
    int i;

    for (i = 0; i < 4; i++)
        fork();
    return EXIT_SUCCESS;
}
```

A.L.2.10. Utilizando el siguiente programa, explicar la salida de la línea A.

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int value = 5;

int main() {
    pid_t pid;
    pid = fork();

    if (pid == 0) { /* proceso hijo */
        value += 15;
        return 0;
    } else if (pid > 0) { /* proceso padre */
        wait(NULL);
        printf("PADRE: valor = %d", value); /* Línea A */
        return EXIT_SUCCESS;
    }
}

```

A.L.2.11. Desarrollar los siguientes ejercicios utilizando el lenguaje de programación C.

- Un proceso debe crear un proceso hijo y esperar por su finalización. El proceso hijo debe ejecutar un bucle en el cual se produzcan diez iteraciones, mostrando el número de iteración por la salida estándar. En particular, en cada iteración el proceso hijo debe suspenderse a sí mismo por un segundo.
- Modificar el ejercicio anterior de forma tal que el proceso padre muestre por la salida estándar tanto su PID como el PID del proceso hijo. Análogamente, el proceso hijo debe mostrar su propio PID.
- Explicar, según tu criterio, qué sucedería si el proceso padre finaliza antes de que lo haga el proceso hijo. Luego, implementá un programa que permita determinar si tu respuesta era correcta.
- Un proceso debe crear 10 procesos hijos. El primer proceso hijo debe suspenderse por 1 segundo, mientras que cada nuevo proceso creado debe suspenderse un segundo después que el proceso previo. Finalmente, el proceso padre debe esperar sincrónicamente en orden decreciente por la finalización de cada proceso hijo, utilizando para dicha tarea el ID otorgado por el kernel en cada paso de la creación. En otras palabras, el proceso padre deberá esperar en primera instancia por la finalización del último proceso hijo creado.
- Ejecutar y monitorear el estado de los procesos creados en el inciso anterior mediante el comando "ps". ¿A qué conclusión se puede llegar?

A.L.2.12. La secuencia de Fibonacci puede expresarse formalmente como:

$$\begin{aligned}
 fib_0 &= 0 \\
 fib_1 &= 1 \\
 fib_n &= fib_{n-1} + fib_{n-2}
 \end{aligned}$$

Siguiendo los lineamientos del ejercicio 10, escribir un programa que compute la secuencia de Fibonacci utilizando un nuevo proceso hijo. La cantidad de términos de la

secuencia debe ser provisto como un parámetro al invocar el programa desde la línea de comandos. El proceso hijo debe mostrar cada término computado por la salida estándar.

A.L.2.13. ¿Qué entendés por POSIX y POSIX Threads (*pthread*)?

A.L.2.14. ¿Qué hace el siguiente programa?

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define TAM 100

void *sacar_no_p(void *arg){
    int *nros_p = (int *) arg;
    int i, j, h;

    for(i = 2; i < TAM; ++i) {
        if(nros_p[i]) {
            for(h = 2; i*h <= TAM; ++h)
                nros_p[i*h] = 0;
        }
    }
    pthread_exit(0);
}

void main(void){
    int i = 0;
    pthread_t hilo_p;
    int *numeros = (int*)malloc((TAM)*sizeof(int));

    numeros[0]=0;
    numeros[1]=0;
    for(i = 2; i < TAM ; i++){numeros[i] = 1;}

    pthread_create(&hilo_p, NULL, sacar_no_p, (void*)numeros);
    pthread_join(hilo_p, NULL);

    for(i = 0; i < TAM ; i++){
        if(numeros[i]){
            printf("%d -", i);
        }
    }
    printf("\n", i);
    free(numeros);
}
```

A.L.2.15. Resolver los siguientes ejercicios utilizando el API POSIX Threads:

- a) Un proceso debe crear n hilos de ejecución, donde n corresponde a un parámetro pasado por la línea de comandos. Cada hilo debe imprimir su identificador y finalizar. El proceso principal debe esperar por la finalización de todos sus hilos de ejecución.

- b) Un proceso debe crear n hilos de ejecución, donde n corresponde a un parámetro pasado por la línea de comandos. Cada hilo debe calcular la sumatoria de todos los números naturales que precedan a un argumento especificado. El primer argumento especificado corresponde a n , mientras que los restantes argumentos son mayores en una unidad a su predecesor. Es decir, el primer hilo recibirá como argumento n , mientras que el hilo k ($k \geq n$) recibirá como argumento $n + k$. Cada hilo debe imprimir la sumatoria computada por la salida estándar. El proceso principal debe esperar por la finalización de todos los hilos creados antes de finalizar.
- c) Repetir el ejercicio anterior, aunque en esta ocasión cada sumatoria computada debe ser mostrada por la salida estándar por el proceso principal, una vez que todos los hilos creados finalicen con su ejecución.
- d) Resolver el ejercicio 11 de forma tal que el proceso principal utilice un nuevo hilo de ejecución para realizar el cálculo de la secuencia de Fibonacci. Una vez que la secuencia haya sido calculada, el proceso principal debe crear un nuevo hilo de ejecución cuyo objetivo sea mostrar por la salida estándar la secuencia calculada.
- e) Un proceso dado consta de m hilos de ejecución. Cada hilo es denominado *worker* dada su naturaleza inherente para realizar trabajo útil por demanda. Cada *worker* puede realizar dos tareas específicas: calcular sumatorias o factoriales. Al crear un nuevo hilo de ejecución, el proceso principal comunica al mismo su identificador: un número entero y la operación que debe realizar. Al finalizar la ejecución de cada hilo, el proceso principal debe mostrar por la salida estándar el identificador del hilo, la operación realizada y el resultado obtenido.



Microsoft Windows

El anexo Microsoft Windows no es obligatorio pero se recomienda la resolución del mismo para tener una mejor comprensión de dicho sistema operativo.

A.W.2.1. Microsoft provee varias herramientas que pueden utilizarse para explorar internamente el Sistema Operativo, las cuales serán necesarias para resolver algunos de los ejercicios del anexo.

- a) Instalación *Debugging Tools for Windows*: ingresar al siguiente sitio y seguir las instrucciones de "*As a standalone tool set*".
 - i) [https://msdn.microsoft.com/en-us/library/windows/hardware/ff551063\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/hardware/ff551063(v=vs.85).aspx)
- b) Instalación *Sysinternals Suite*: para descargarla ingresar al siguiente sitio y en la sección "*Utilities*" elegir "*Sysinternals Suite*".
 - i) <https://technet.microsoft.com/en-us/sysinternals/default.aspx>

A.W.2.2. En Microsoft Windows cada proceso apunta a su padre o proceso creador. Si el padre no existe, ésta información no se actualiza. Por lo tanto, es posible que un proceso tenga una referencia a un padre inexistente. Esto no constituye un problema, dado que ningún componente en el sistema confía en que esta información se mantenga actualizada. Este comportamiento puede analizarse de la siguiente manera:

- a) La herramienta Tlist.exe disponible en las "*Debugging Tools for Windows*" muestra el árbol de procesos con la opción /t, es decir tlist /t. Al ejecutarla, comprobar que los procesos cuyos padres no están vivos estén justificados a la izquierda (por ejemplo: "explorer.exe"). Si existiera un proceso abuelo, no hay manera de encontrar esa relación, dado que Windows mantiene sólo el ID del proceso creador y no un enlace de vuelta al creador del creador.
- b) Comprobar que la herramienta *Process Explorer* ("procexp.exe") de "*Sysinternals Suite*" muestra la misma información.

A.W.2.3. Para demostrar que Windows mantiene solo el ID del proceso padre (ver A.W.2.2), realizar los siguientes pasos:

- 1) Ejecutar una consola CMD.
- 2) Escribir "title Padre", para cambiar el título de la ventana a "Padre".
- 3) Escribir "start cmd", la cual ejecuta una segunda consola CMD.
- 4) En la segunda consola CMD, escribir "title Hijo".
- 5) Ejecutar el "Task Manager".
- 6) En la segunda consola CMD, ejecutar "mspaint".
- 7) Regresar a la segunda consola CMD y escribir "exit" (notarás que el paint permanece).
- 8) En el "Task Manager" hacer click en el tab *Details*.
- 9) En el "Task Manager" encontrar "cmd.exe", hacer click derecho y seleccionar "End process tree"; confirmar la ventana.

- 10) La primer consola desaparecerá pero aún deberás ver la ventana de *Paint*, dado que es nieto del proceso CMD que finalizó y que el proceso intermedio (el padre de *Paint*) también finalizó, motivo por el cual no hay relación entre el padre y el nieto.

A.W.2.4. Contrastar los dos puntos anterior con GNU/Linux.

A.W.2.5. ¿Qué es un servicio de Windows?

A.W.2.6. Desarrollar los siguientes ejercicios utilizando *PowerShell*:

- a) Obtener todos los procesos ordenados por id.
- b) Obtener todos los procesos ordenados por uso de CPU.
- c) Obtener todos los servicios.
- d) Haciendo uso del *pipe*, obtener la cantidad de servicios.
- e) Haciendo uso del *pipe*, obtener la cantidad de procesos.

A.W.2.7. Descargar la herramienta *Visual Studio Community* de la página oficial de Microsoft. Instalarla para poder desarrollar una aplicación en el lenguaje de programación C++. A continuación, ejecutar el siguiente código provisto como ejemplo para crear un proceso:
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms682512\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682512(v=vs.85).aspx)