



# Instituto Politécnico Nacional

Escuela Superior de Cómputo

---

*Sistemas Operativos*

## *“Práctica 3. Administrador de procesos en Linux y Windows (1)”*

**Grupo:** 2CM9

**Integrantes:**

- Martínez Coronel Brayan Yosafat.
- Monteros Cervantes Miguel Angel.
- Ramírez Olvera Guillermo.
- Sánchez Méndez Edmundo Josue.

**Profesor:** Cortés Galicia Jorge



## **Práctica 3. Administrador de procesos en Linux y Windows (1)**

### **Introducción**

Un proceso podría ser una instancia de un programa en ejecución. A los procesos frecuentemente se les refiere como tareas. El contexto de un programa que está en ejecución es lo que se llama un proceso. Por ejemplo, Linux es un sistema operativo multitarea y multiusuario. Esto quiere decir que múltiples procesos pueden operar simultáneamente sin interferirse unos con los otros. Cada proceso tiene la "ilusión" que es el único proceso en el sistema y que tiene acceso exclusivo a todos los servicios del sistema operativo.

Programas y procesos son entidades distintas, múltiples instancias de un programa pueden ejecutarse simultáneamente. Cada instancia es un proceso separado. Por ejemplo, si usuarios desde equipos diferentes, ejecutan el mismo programa al mismo tiempo, habría tantas instancias del mismo programa, es decir, procesos distintos.

Cada proceso que se inicia es referenciado con un número de identificación único conocido como Process ID PID, que es siempre un entero positivo, al final veremos una curiosidad y diferencia entre Linux y Windows. Prácticamente todo lo que se está ejecutando en el sistema en cualquier momento es un proceso, incluyendo el shell, el ambiente gráfico que puede tener múltiples procesos, etc. La excepción a lo anterior es el kernel en sí, el cual es un conjunto de rutinas que residen en memoria y a los cuales los procesos a través de llamadas al sistema pueden tener acceso.

Ademas, como hemos visto en las lecturas dejados, nos damos cuenta de que es de nuestro interés tener un buen proceso de planificación ya que se pretende usar la CPU el mayor tiempo posible, y esto se debe de hacer con base a los estados de los procesos, en espera, preparado y terminado, mas adelante veremos como en podemos saber el estado de procesos en la computadora (Ubuntu) y de procesos que nosotros creemos y ademas de saber sus estados poder tener control sobre los mismos.

Ademas de ver Planificadores y Multiprogramación de una forma práctica, el primer punto es el programa con 6 procesos con sustitución de código en donde llevaremos una planificación de pasos a ejecutar y el de multiprogramación con 6 procesos con copia exacta de código y veremos un caso en donde nos encontraremos en problemas si no tenemos cuidado en como usaremos los procesos.

Concluyendo esta introducción hay que mencionar que los comandos y funciones que se verán más adelante, también son llamadas del sistema, por lo que, nuestro programa seguirá teniendo las ventajas que en la practica de la anterior unidad se comentaron.

## 1. Competencias.

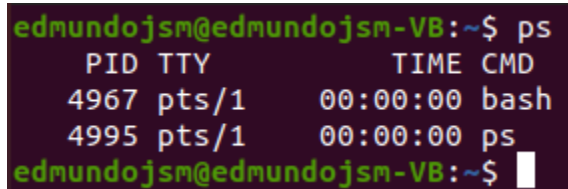
El alumno aprende a familiarizarse con el administrador de procesos del sistema operativo Linux y Windows a través de la creación de nuevos procesos por copia exacta de código y/o por sustitución de código para el desarrollo de aplicaciones concurrentes sencillas.

## 2. Desarrollo.

### 2.1. Sección Linux

#### 2.1.1. Ejecución de los comandos *ps* y *ps -fea*

- Comando **ps**. Muestra el número del proceso (**PID**), tipo de terminal (**TTY**), tiempo acumulado de ejecución (**TIME**) y el nombre del comando (**CMD**).



```
edmundojm@edmundojm-VB:~$ ps
  PID TTY          TIME CMD
 4967 pts/1        00:00:00 bash
 4995 pts/1        00:00:00 ps
edmundojm@edmundojm-VB:~$
```

- Comando **ps -fea**. Muestra el usuario propietario del proceso, la identificación del proceso, el padre de proceso, porcentaje de la CPU utilizado por el proceso, hora de inicio del proceso, terminal asociada al proceso (si no hay terminal aparece entonces un '?'), tiempo de uso de CPU acumulado por el proceso y el nombre del programa o comandó que inicio el proceso.

```

edmundojm@edmundojm-VB:~$ ps -fea
UID          PID    PPID    C  TIME TTY          TIME CMD
root           1        0  0  18:12 ?        00:00:07 /sbin/init splash
root           2        0  0  18:12 ?        00:00:00 [kthreadd]
root           3        2  0  18:12 ?        00:00:00 [rcu_gp]
root           4        2  0  18:12 ?        00:00:00 [rcu_par_gp]
root           6        2  0  18:12 ?        00:00:00 [kworker/0:0H-kblockd]
root           9        2  0  18:12 ?        00:00:00 [mm_percpu_wq]
root          10        2  0  18:12 ?        00:00:00 [ksoftirqd/0]
root          11        2  0  18:12 ?        00:00:01 [rcu_sched]
root          12        2  0  18:12 ?        00:00:00 [migration/0]
root          13        2  0  18:12 ?        00:00:00 [idle_inject/0]
root          14        2  0  18:12 ?        00:00:00 [cpuhp/0]
root          15        2  0  18:12 ?        00:00:00 [kdevtmpfs]
root          16        2  0  18:12 ?        00:00:00 [netns]
root          17        2  0  18:12 ?        00:00:00 [rcu_tasks_kthre]
root          18        2  0  18:12 ?        00:00:00 [kauditd]
root          19        2  0  18:12 ?        00:00:00 [khungtaskd]
root          20        2  0  18:12 ?        00:00:00 [oom_reaper]
root          21        2  0  18:12 ?        00:00:00 [writeback]
root          22        2  0  18:12 ?        00:00:00 [kcompactd0]
root          23        2  0  18:12 ?        00:00:00 [ksmd]
root          24        2  0  18:12 ?        00:00:00 [khugepaged]
root          70        2  0  18:12 ?        00:00:00 [kintegrityd]
root          71        2  0  18:12 ?        00:00:00 [kblockd]
root          72        2  0  18:12 ?        00:00:00 [blkcg_punt_bio]
root          73        2  0  18:12 ?        00:00:00 [tpm_dev_wq]
root          74        2  0  18:12 ?        00:00:00 [ata_sff]
root          75        2  0  18:12 ?        00:00:00 [md]
root          76        2  0  18:12 ?        00:00:00 [edac-poller]
root          77        2  0  18:12 ?        00:00:00 [devfreq_wq]
root          78        2  0  18:12 ?        00:00:00 [watchdogd]
root          83        2  0  18:12 ?        00:00:00 [kswapd0]
root          84        2  0  18:12 ?        00:00:00 [ecryptfs-kthrea]
root          86        2  0  18:12 ?        00:00:00 [kthrotld]
root          87        2  0  18:12 ?        00:00:00 [acpi_thermal_pm]
root          88        2  0  18:12 ?        00:00:00 [scsi_eh_0]
root          89        2  0  18:12 ?        00:00:00 [scsi_tmf_0]
root          90        2  0  18:12 ?        00:00:00 [scsi_eh_1]
root          91        2  0  18:12 ?        00:00:00 [scsi_tmf_1]
root          93        2  0  18:12 ?        00:00:00 [vfio-irqfd-clea]
root          95        2  0  18:12 ?        00:00:00 [ip6_addrconf]
root         104        2  0  18:12 ?        00:00:00 [kstrp]
root         107        2  0  18:12 ?        00:00:00 [kworker/u3:0]
root         120        2  0  18:12 ?        00:00:00 [charger_manager]
root         121        2  0  18:12 ?        00:00:00 [kworker/0:1H-kblockd]

```

```

root         166        2  0  18:12 ?        00:00:00 [scsi_eh_2]
root         167        2  0  18:12 ?        00:00:00 [scsi_tmf_2]
root         189        2  0  18:12 ?        00:00:00 [jbd2/sda5-8]
root         198        2  0  18:12 ?        00:00:00 [ext4-rsv-conver]
root         229        1  0  18:12 ?        00:00:00 /lib/systemd/systemd-journald
root         254        2  0  18:12 ?        00:00:00 [irq/18-vmwgfx]
root         255        2  0  18:12 ?        00:00:00 [ttm_swap]
root         258        2  0  18:12 ?        00:00:00 [loop0]
root         259        2  0  18:12 ?        00:00:00 [loop1]
root         261        2  0  18:12 ?        00:00:00 [loop2]
root         263        2  0  18:12 ?        00:00:00 [loop3]
root         265        2  0  18:12 ?        00:00:00 [loop4]
root         267        2  0  18:12 ?        00:00:00 [loop5]
root         269        2  0  18:12 ?        00:00:00 [loop6]
root         271        2  0  18:12 ?        00:00:00 [loop7]
root         275        1  0  18:12 ?        00:00:02 /lib/systemd/systemd-udevd
root         279        2  0  18:12 ?        00:00:00 [loop9]
root         312        2  0  18:12 ?        00:00:00 [lpri-VBoxWQueue]
root         331        2  0  18:12 ?        00:00:00 [cryptd]
systemd+     499        1  0  18:12 ?        00:00:00 /lib/systemd/systemd-resolved
systemd+     500        1  0  18:12 ?        00:00:00 /lib/systemd/systemd-timesyncd
root         533        1  0  18:12 ?        00:00:00 /usr/lib/accounts-service/accounts-daemon
root         534        1  0  18:12 ?        00:00:00 /usr/sbin/acpid
avahi        537        1  0  18:12 ?        00:00:00 avahi-daemon: running [edmundojm-VB.local]
root         538        1  0  18:12 ?        00:00:00 /usr/sbin/cron -f
message+     540        1  0  18:12 ?        00:00:03 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog
root         541        1  0  18:12 ?        00:00:00 /usr/sbin/NetworkManager --no-daemon
root         553        1  0  18:12 ?        00:00:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
root         558        1  0  18:12 ?        00:00:01 /usr/lib/policykit-1/polkitd --no-debug
syslog       562        1  0  18:12 ?        00:00:00 /usr/sbin/rsyslogd -n -iNONE
root         578        1  0  18:12 ?        00:00:00 /usr/libexec/switcheroo-control
root         572        1  0  18:12 ?        00:00:00 /lib/systemd/systemd-logind
root         576        1  0  18:12 ?        00:00:00 /usr/lib/udisks2/udisksd
root         577        1  0  18:12 ?        00:00:00 /sbin/wpa_supplicant -u -s -O /run/wpa_supplicant
avahi        584        537  0  18:12 ?        00:00:00 avahi-daemon: chroot helper
root         629        1  0  18:12 ?        00:00:00 /usr/sbin/cupsd -l
root         665        1  0  18:12 ?        00:00:00 /usr/sbin/cups-browsed
root         670        1  0  18:12 ?        00:00:00 /usr/sbin/ModemManager --filter-policy=strict
root         677        1  0  18:12 ?        00:00:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
root         696        1  0  18:12 ?        00:00:00 /usr/sbin/gdm3
root         711        696  0  18:12 ?        00:00:00 gdm-session-worker [pam/gdm-autologin]
whoopsie     732        1  0  18:12 ?        00:00:00 /usr/bin/whoopsie -f
kernoops     736        1  0  18:12 ?        00:00:00 /usr/sbin/kerneloops --test
kernoops     738        1  0  18:12 ?        00:00:00 /usr/sbin/kerneloops
edmundojm+   803        1  0  18:12 ?        00:00:01 /lib/systemd/systemd --user
edmundojm+   804        803  0  18:12 ?        00:00:00 (sd-pan)

```

```

edmund+ 809 803 0 18:12 ? 00:00:00 /usr/bin/pulseaudio --daemonize=no --log-target=journal
edmund+ 812 803 0 18:12 ? 00:00:00 /usr/libexec/tracker-miner-fs
edmund+ 814 1 0 18:12 ? 00:00:00 /usr/bin/gnome-keyring-daemon --daemonize --login
edmund+ 818 711 0 18:12 tty2 00:00:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-sessio
edmund+ 820 818 0 18:12 tty2 00:00:04 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset
edmund+ 821 1 0 18:12 ? 00:00:00 /usr/libexec/rtkit-daemon
edmund+ 825 803 0 18:12 ? 00:00:01 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-activation --sysl
edmund+ 830 803 0 18:12 ? 00:00:00 /usr/libexec/gvfsd
edmund+ 835 803 0 18:12 ? 00:00:00 /usr/libexec/gvfsd-fuse /run/user/1000/gvfs -f -o big_writes
edmund+ 844 803 0 18:12 ? 00:00:00 /usr/libexec/gvfs-udisks2-volume-monitor
edmund+ 861 803 0 18:12 ? 00:00:00 /usr/libexec/gvfs-mtp-volume-monitor
edmund+ 865 803 0 18:12 ? 00:00:00 /usr/libexec/gvfs-goa-volume-monitor
edmund+ 869 803 0 18:12 ? 00:00:00 /usr/libexec/goa-daemon
edmund+ 884 803 0 18:13 ? 00:00:00 /usr/libexec/goa-identity-service
edmund+ 889 803 0 18:13 ? 00:00:00 /usr/libexec/gvfs-afc-volume-monitor
edmund+ 895 803 0 18:13 ? 00:00:00 /usr/libexec/gvfs-gphoto2-volume-monitor
edmund+ 901 1 0 18:13 ? 00:00:00 /usr/lib/upower/upowerd
edmund+ 906 818 0 18:13 tty2 00:00:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
edmund+ 988 906 0 18:13 ? 00:00:00 /usr/bin/ssh-agent /usr/bin/im-launch env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session -
edmund+ 1007 803 0 18:13 ? 00:00:00 /usr/libexec/at-spi-bus-launcher
edmund+ 1013 1007 0 18:13 ? 00:00:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print
edmund+ 1018 803 0 18:13 ? 00:00:00 /usr/libexec/gnome-session-ctl --monitor
edmund+ 1026 803 0 18:13 ? 00:00:00 /usr/libexec/gnome-session-binary --systemd-service --session=ubuntu
edmund+ 1059 803 1 18:13 ? 00:00:35 /usr/bin/gnome-shell
edmund+ 1148 2 0 18:13 ? 00:00:00 [loop10]
edmund+ 1160 1059 0 18:13 ? 00:00:00 ibus-daemon --panel disable --xim
edmund+ 1164 1160 0 18:13 ? 00:00:00 /usr/libexec/ibus-memconf
edmund+ 1165 1160 0 18:13 ? 00:00:04 /usr/libexec/ibus-extension-gtk3
edmund+ 1168 803 0 18:13 ? 00:00:00 /usr/libexec/ibus-x11 --kill-daemon
edmund+ 1171 803 0 18:13 ? 00:00:00 /usr/libexec/ibus-portal
edmund+ 1181 803 0 18:13 ? 00:00:00 /usr/libexec/at-spi2-registrd --use-gnome-session
edmund+ 1202 1 0 18:13 ? 00:00:03 /usr/lib/snapd/snapd
edmund+ 1211 803 0 18:13 ? 00:00:00 /usr/libexec/xdg-permission-store
edmund+ 1216 803 0 18:13 ? 00:00:00 /usr/libexec/gnome-shell-calendar-server
edmund+ 1222 803 0 18:13 ? 00:00:00 /usr/libexec/evolution-source-registry
edmund+ 1232 803 0 18:13 ? 00:00:00 /usr/libexec/evolution-calendar-factory
edmund+ 1244 803 0 18:13 ? 00:00:00 /usr/libexec/dconf-service
edmund+ 1249 803 0 18:13 ? 00:00:00 /usr/libexec/evolution-addressbook-factory
edmund+ 1321 803 0 18:13 ? 00:00:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.Shell.Notifications
edmund+ 1323 830 0 18:13 ? 00:00:00 /usr/libexec/gvfsd-trash --spawner :1.4 /org/gtk/gvfs/exec_spaw/0
edmund+ 1340 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-all-ty-settings
edmund+ 1342 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-color
edmund+ 1345 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-datetime
edmund+ 1347 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-housekeeping
edmund+ 1349 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-keyboard
edmund+ 1351 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-media-keys
edmund+ 1355 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-power
edmund+ 1356 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-print-notifications
edmund+ 1358 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-rfkill
edmund+ 1359 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-screensaver-proxy
edmund+ 1361 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-sharing
edmund+ 1362 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-smartcard
edmund+ 1366 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-sound
edmund+ 1373 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-usb-protection
edmund+ 1374 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-wacom
edmund+ 1379 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-wwan
edmund+ 1381 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-xsettings
edmund+ 1392 1026 0 18:13 ? 00:00:00 /usr/libexec/evolution-data-server/evolution-alarm-notify
edmund+ 1401 1026 0 18:13 ? 00:00:00 /usr/libexec/gsd-disk-utility-notify
edmund+ 1458 803 0 18:13 ? 00:00:00 /usr/libexec/gsd-printer
edmund+ 1462 1160 0 18:13 ? 00:00:00 /usr/libexec/ibus-engine-simple
edmund+ 1471 1 0 18:13 ? 00:00:00 /usr/libexec/colord
edmund+ 1643 803 0 18:14 ? 00:00:00 /usr/libexec/gvfsd-metadata
edmund+ 1646 1026 0 18:14 ? 00:00:00 update-notifier
edmund+ 2041 2 0 18:27 ? 00:00:00 [kworker/0:1-events]
edmund+ 2112 2 0 18:30 ? 00:00:00 [kworker/u2:0-events_power_efficient]
edmund+ 4901 803 0 18:38 ? 00:00:00 /usr/bin/seahorse --gapplication-service
edmund+ 4903 803 0 18:38 ? 00:00:00 /usr/bin/gnome-calendar --gapplication-service
edmund+ 4905 803 0 18:38 ? 00:00:00 /usr/libexec/gnome-terminal-server
edmund+ 4967 4905 0 18:38 pts/1 00:00:00 bash
edmund+ 4997 2 0 18:39 ? 00:00:00 [kworker/u2:2-events_power_efficient]
edmund+ 5018 2 0 18:39 ? 00:00:00 [kworker/0:0-cgroup_destroy]
edmund+ 5048 2 0 18:45 ? 00:00:00 [kworker/u2:1-events_unbound]
edmund+ 5111 2 0 18:45 ? 00:00:00 [kworker/0:2-events]
edmund+ 5116 2 0 18:45 ? 00:00:00 [kworker/0:3-events]
edmund+ 5117 2 0 18:45 ? 00:00:00 [kworker/0:4]
edmund+ 5122 4967 0 18:48 pts/1 00:00:00 ps -fea
edmund+jsm@edmund+jsm-VB:--$

```

## 2.1.2. Información de comandos y llamadas al sistema

Información del comando **ps**:

- **ps**: Muestra un árbol de procesos.
- **ps -aux** o **texttt ps aux**: Lista de los programas que se están ejecutando actualmente con su *PID*.

- **ps -ef | grep nombre-proceso:** Lista los procesos que se están ejecutando con que contengan la cadena nombre-proceso.
- **ps aux:** Lista los procesos con la información de *USER, PID, CPU, MEM, VSZ, RSS, TTY, STAT, START, TIME, COMMAND*.
- **ps ax:** Lista los procesos con la información de *PID, TTY, STAT, TIME, COMMAND*.
- **ps -a:** Lista los procesos de todos los usuarios.
- **ps -u:** Lista información del proceso como por ejemplo el usuario que lo está corriendo, la utilización de CPU y memoria, etc.
- **ps -x:** Lista procesos de todas las terminales y usuarios
- **ps -l:** Muestra información que incluye el *UID*.
- **ps -e:** Visualiza información sobre “todos” los procesos del sistema.
- **ps axjf:** Mostrará un árbol jerárquico con la ruta del programa al que pertenece el proceso.
- **ps aux | grep bash:** Realiza filtrado sobre ps para obtener únicamente los procesos pertenecientes a *bash*.

Ahora, veremos los campos de la salida del comando **ps**:

- **UID:** ID de usuario.
- **PID:** ID del proceso.
- **PPID:** ID del proceso padre.
- **PGID:** ID de grupo de un proceso.
- **PRI:** Prioridad del proceso.
- **NI:** valor de bondad, más elevado menor prioridad.
- **VSZ:** Tamaño de la memoria virtual del proceso en Kb.
- **RSS:** Tamaño de la memoria física usada en Kb.
- **WCHAN.** para los procesos que esperan o dormidos, enumera el evento que espera.
- **STAT:** Estado del proceso:
  - R: Ejecutable.
  - D: Interrumpió.
  - S: Suspendido.
  - s: Es el proceso líder de la sesión.

- T: Detenido.
- Z: Zombie.
- X: Muerto.
- <: Tiene una prioridad alta que lo normal.
- N: Tiene una prioridad menor que lo normal.
- L: Tiene paginas bloqueadas en la memoria (para E/S en tiempo real y personalizadas).
- I: Es multiproceso.
- +: Está en el grupo de proceso de primer plano,
- TTY: nombre de la terminal a la que está asociado al proceso.
- TIME: tiempo que lleva en ejecución.

Información del comando **fork()**: Llamada para la creación de un proceso por copia exacta de código. Valores de retorno:

- 0 si es el proceso hijo.
- -1 si no se ha podido crear el proceso hijo
- Mayor a cero si es el proceso padre

Para hacer uso de esta llamada se deben incluir en la cabecera:

- **sys/types.h**
- **unistd.h**

Información del comando **execv (const char\* path, char\* const argv[ ])**: Llamada para la creación de un proceso por sustitución de código. Si la llamada falla se retornara -1. Parámetros:

- **const char\* path**: Ruta del nuevo programa a ejecutar con su ruta.
- **char\* const argv[ ]**: Lista de argumentos disponibles para el nuevo programa. El último de los punteros debe ser NULL. Por convención, este array debe contener al menos un elemento (nombre del programa).

Para hacer uso de esta llamada se deben incluir en la cabecera **unistd.h**

Información del comando **getpid()**: Llamada al sistema para obtener el identificador del proceso quien invoca la llamada. Retorna un entero del tipo **pid\_t**. Para hacer uso de esta llamada se deben incluir en la cabecera:

- **sys/types.h**
- **unistd.h**

Información del comando **getppid()**: Llamada al sistema para obtener el identificador del proceso padre del proceso quien invoca la llamada. Retorna un entero del tipo **pid\_t**. Para hacer uso de esta llamada se deben incluir en la cabecera:

- **sys/types.h**
- **unistd.h**

Información del comando **wait(int \*status)**: Llamada al sistema para hacer esperar un proceso padre a uno de sus procesos hijo. Parámetro:

- **int \*status**: Esta función recibe como argumento un puntero a una variable entera en la que se colocara el estado actual del proceso hijo y retorna el **pid** del proceso hijo que termino. El argumento también puede colocarse **NULL**.

Funciones similares a **execv()**

La familia **exec...()** es un conjunto de funciones que en esencia realizan la misma actividad ya que solo difieren en la forma de pasar sus argumentos, son utilizadas para poner en ejecución un proceso determinado, la característica es que las instrucciones del proceso que las invoca son sustituidas por las instrucciones del proceso indicado. Si la llamada falla se retornara -1. Las funciones son:

- **int execl (char \*path, char \*arg0, char \*arg1, . . . ,char \*argN, char \*null)**
- **int execl (char \*path, char \*arg0, ... ,char \*argN, char \*null, char \*envp[])**
- **int execlp (char \*file, char \*arg0, char \*arg1, ... ,char \*argN, char \*null)**
- **int execv (char \*path, char \*argv[])**
- **int execve (char \*path, char \*argv[], char \*envp[])**
- **int execvp (char \*file, char \*argv[])**

Características de los parámetros:

- **path, file**: Nombre del nuevo programa a ejecutar con su ruta. Las versiones de **exec** que utilizan **file** en lugar de **path** utilizan la variable de entorno **PATH** para localizar el programa a ejecutar, por lo que en esos casos no es necesario especificar la ruta al programa si este se encuentra en alguno de los directorios especificados en **PATH**.
- **arg0**: Primer argumento del programa. Por convención suele asignarse el nombre del programa sin la ruta.
- **arg1, ..., argN, NULL**: Conjunto de parámetros que recibe el programa para su ejecución.
- **argv**: Lista de argumentos disponibles para el nuevo programa. El último de los punteros debe ser **NULL**. Por convención, este array debe contener al menos un elemento (nombre del programa).
- **envp**: Cadena de caracteres constituyen el entorno de ejecución del nuevo programa.



Para hacer uso de esta llamada se debe incluir la cabecera **unistd.h**

### 2.1.3. Ejemplo de creación de procesos por copia exacta de código

Código(CopiaExacta.c)

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <stdlib.h>
04.
05. int main(void){
06.     int id_proc;
07.     id_proc=fork();
08.     if(id_proc == 0){
09.         printf("Soy el proceso hijo\n");
10.         exit(0);
11.     }else{
12.         printf("Soy el proceso padre\n");
13.         exit(0);
14.     }
15. }
```

```
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$ gcc CopiaExacta.c
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$ ./a.out
Soy el proceso padre
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$ Soy el proceso hijo
```

Ejecuta internamente de manera concurrente dos procesos mediante la invocación de la llamada al sistema `fork()`. El sistema en el que se ejecutó el código está configurado de la manera en que le da prioridad al proceso padre la mayoría de las veces. Por medio de los `if` y el valor que retorna **fork()** podemos ver la salida de los `printf` en pantalla y observar ambos procesos (los cuales finalizan con la llamada al sistema `exit (0)`).

Código(CopiaExacta1.c)

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <stdlib.h>
04.
05. int main(void){
06.     int id_proc;
07.     id_proc=fork();
08.     if(id_proc == 0){
09.         printf("Soy el proceso hijo\n");
10.     }else{
11.         printf("Soy el proceso padre\n");
12.     }
13.     printf("Mensaje en ambos\n");
14.     exit(0);
15. }
```

```
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$ gcc CopiaExacta1.c
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$ ./a.out
Soy el proceso padre
Mensaje en ambos
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$ Soy el proceso hijo
Mensaje en ambos
```

Ejecuta internamente de manera concurrente dos procesos mediante la invocación de la llamada al sistema **fork()**. El sistema en el que se ejecutó el código está configurado de manera que le da prioridad al proceso padre. Por medio de los `if` y el valor que retorna **fork()** podemos ver la salida de los `printf` en pantalla y

observar ambos procesos que finalizan con la llamada al sistema `exit(0)` la cual tiene un `printf("Mensaje en ambos")` y si se colocara antes el `exit(0)` no lo imprimiría en pantalla.

#### 2.1.4. Creación del árbol de procesos

Para este punto se decidió crear una estructura **nodo** la cual se encuentra en el código "EstructuraArbol.h", la cual simplemente contiene un arreglo de sus nodos hijos. Después se crea de manera recursiva (en profundidad, usando la técnica de DFS) el árbol de procesos de acuerdo con lo que contenga el nodo raíz.

Código(EstructuraArbol.h)

```
01. typedef struct nodo{
02.     int numhijos;
03.     struct nodo** hijos;
04. } nodo;
05.
06. nodo* nuevoNodo(){
07.     nodo *tmp = calloc(1, sizeof(nodo*));
08.     tmp->hijos = calloc(50, sizeof(nodo*));
09.     return tmp;
10. }
```

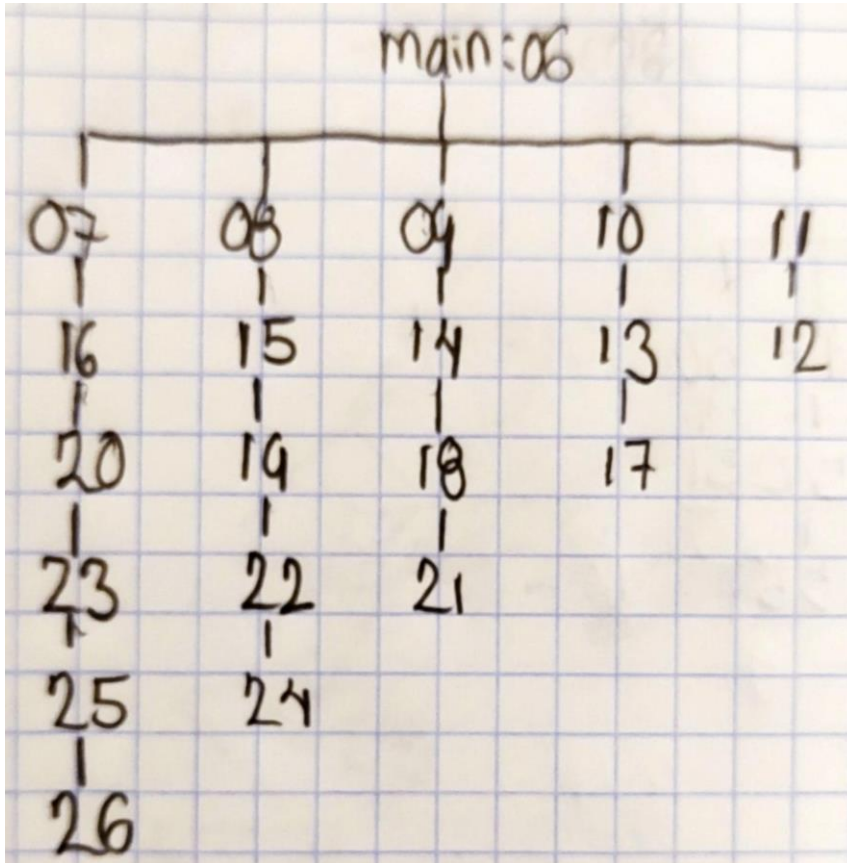
Código(ArbolProcesos.c)

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <stdlib.h>
04. #include <sys/wait.h>
05. #include "EstructuraArbol.h"
06.
07. void imprimirEspaciado(int n){
08.     for(int i = 0; i < n; ++i) printf(".");
09. }
10.
11. void arbolProcesos(nodo* pos, int nivel){
12.     if(!pos) return;
13.     if(nivel == 0) printf("main(): %d\n", getpid());
14.     int id_proc;
15.     for(int i = 0; i < pos->numhijos; ++i){
16.         id_proc = fork();
17.         if(id_proc == 0){ //Hijo, entonces continua recursion
18.             imprimirEspaciado(nivel + 1);
19.             printf("Inicio proceso hijo %d\n", getpid());
20.             arbolProcesos(pos->hijos[i], nivel + 1);
21.             exit(0);
22.         }else{ //Padre, entonces esta en su mismo nivel
23.             imprimirEspaciado(nivel);
24.             printf("Creando proceso hijo %d, padre: %d\n", id_proc, getpid());
25.         }
26.     }
27.     for(int i = 0; i < pos->numhijos; ++i){
28.         int id = wait(0);
29.         imprimirEspaciado(nivel + 1);
30.         printf("Fin proceso hijo %d\n", id);
31.     }
32. }
33.
34. int main(){
35.     nodo* raiz = nuevoNodo();
36.     raiz->numhijos = 5;
37.     nodo** principal = raiz->hijos;
38.     for(int i = 0; i < 5; ++i){
39.         principal[i] = nuevoNodo();
40.         int abajo = 6-i;
41.         nodo* pos = principal[i];
42.         for(int j = 1; j < abajo; ++j){
43.             pos->numhijos = 1;
44.             pos->hijos[0] = nuevoNodo();
45.             pos = pos->hijos[0];
46.         }
47.     }
48.     arbolProcesos(raiz, 0);
49.     return 0;
50. }
```

Resultado al momento de ejecutar y compilar el programa:

```
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$ gcc ArbolProcesos.c
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$ ./a.out
main(): 3006
Creando proceso hijo 3007, padre: 3006
Creando proceso hijo 3008, padre: 3006
Creando proceso hijo 3009, padre: 3006
Creando proceso hijo 3010, padre: 3006
Creando proceso hijo 3011, padre: 3006
.Inicio proceso hijo 3011
..Creando proceso hijo 3012, padre: 3011
..Inicio proceso hijo 3010
..Creando proceso hijo 3013, padre: 3010
..Inicio proceso hijo 3009
..Creando proceso hijo 3014, padre: 3009
..Inicio proceso hijo 3008
..Creando proceso hijo 3015, padre: 3008
..Inicio proceso hijo 3007
..Creando proceso hijo 3016, padre: 3007
...Inicio proceso hijo 3014
...Inicio proceso hijo 3013
...Inicio proceso hijo 3012
...Fin proceso hijo 3012
..Fin proceso hijo 3011
..Creando proceso hijo 3017, padre: 3013
..Creando proceso hijo 3018, padre: 3014
..Inicio proceso hijo 3015
..Creando proceso hijo 3019, padre: 3015
..Inicio proceso hijo 3016
..Creando proceso hijo 3020, padre: 3016
...Inicio proceso hijo 3017
...Fin proceso hijo 3017
...Fin proceso hijo 3013
..Fin proceso hijo 3010
...Inicio proceso hijo 3018
...Creando proceso hijo 3021, padre: 3018
...Inicio proceso hijo 3019
...Creando proceso hijo 3022, padre: 3019
...Inicio proceso hijo 3020
...Creando proceso hijo 3023, padre: 3020
....Inicio proceso hijo 3021
....Fin proceso hijo 3021
...Fin proceso hijo 3018
..Fin proceso hijo 3014
..Fin proceso hijo 3009
....Inicio proceso hijo 3022
....Creando proceso hijo 3024, padre: 3022
....Inicio proceso hijo 3023
....Creando proceso hijo 3025, padre: 3023
.....Inicio proceso hijo 3024
.....Fin proceso hijo 3024
....Fin proceso hijo 3022
...Fin proceso hijo 3019
..Fin proceso hijo 3015
..Fin proceso hijo 3008
.....Inicio proceso hijo 3025
.....Creando proceso hijo 3026, padre: 3025
.....Inicio proceso hijo 3026
.....Fin proceso hijo 3026
.....Fin proceso hijo 3025
....Fin proceso hijo 3023
...Fin proceso hijo 3020
..Fin proceso hijo 3016
..Fin proceso hijo 3007
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$
```

Y de acuerdo con los PID's impresos, el árbol que se creó realmente fue (solo usamos los dos últimos dígitos del PID, pues todos comienzan con 30):



El cual coincide con el árbol solicitado.

#### **2.1.5. Aplicación con seis procesos por copia exacta de código (Matrices)**

En el siguiente archivo se implementa las operaciones básicas entre dos matrices de un tamaño fijo de 7x7 (Se define una variable N la cual puede ser modificada por otro número entero positivo), así como funciones para importar/exportar desde un archivo, imprimir en pantalla y generar matrices aleatorias; con el fin de reutilizar código y que este se vea más presentable. Las entradas de la matriz son del tipo int (entero).

Código(EstructuraMatriz.h)

```

01. #include <stdbool.h>
02. #include <math.h>
03. typedef double* Vector;
04. typedef Vector* Matriz;
05. #define N 7
06.
07. Matriz NuevaMatriz(){
08.     int M = N;
09.     if(N&1)
10.         M+=1;
11.     Matriz A = calloc(M, sizeof(Vector));
12.     for(int i = 0; i < N; ++i)
13.         A[i] = calloc(M, sizeof(double));
14.     return A;
15. }
16.
17. bool esCero(double x){
18.     return fabs(x) < 1e-8;
19. }
20.
21. Matriz Suma(Matriz A, Matriz B){
22.     Matriz C = NuevaMatriz();
23.     for(int i = 0; i < N; ++i)
24.         for(int j = 0; j < N; ++j)
25.             C[i][j] = A[i][j] + B[i][j];
26.     return C;
27. }
28.
29. Matriz Resta(Matriz A, Matriz B){
30.     Matriz C = NuevaMatriz();
31.     for(int i = 0; i < N; ++i)
32.         for(int j = 0; j < N; ++j)
33.             C[i][j] = A[i][j] - B[i][j];
34.     return C;
35. }
36.
37. Matriz Multiplicacion(Matriz A, Matriz B){
38.     Matriz C = NuevaMatriz();
39.     for(int i = 0; i < N; ++i)
40.         for(int j = 0; j < N; ++j)
41.             for(int k = 0; k < N; ++k)
42.                 C[i][j] += A[i][k] * B[k][j];
43.     return C;
44. }
45.
46. Matriz Transpuesta(Matriz A){
47.     Matriz C = NuevaMatriz();
48.     for(int i = 0; i < N; ++i)
49.         for(int j = 0; j < N; ++j)
50.             C[i][j] = A[j][i];
51.     return C;
52. }
53.
54. Matriz Inversa(Matriz A){
55.     Matriz inv = NuevaMatriz();
56.     for(int i = 0; i < N; ++i)
57.         inv[i][i] = 1;
58.
59.     int i = 0, j = 0;
60.     while(i < N && j < N){
61.         if(esCero(A[i][j])){
62.             for(int k = i + 1; k < N; ++k){
63.                 if(!esCero(A[k][j])){
64.                     Vector tmp = A[i];
65.                     A[i] = A[k];
66.                     A[k] = tmp;
67.                     tmp = inv[i];
68.                     inv[i] = inv[k];
69.                     inv[k] = tmp;
70.                     break;
71.                 }
72.             }
73.             if(!esCero(A[i][j])){
74.                 for(int l = 0; l < N; ++l)
75.                     inv[i][l] /= A[i][j];
76.                 for(int l = N - 1; l >= j; --l)
77.                     A[i][l] /= A[i][j];
78.                 for(int k = 0; k < N; ++k){
79.                     if(i == k) continue;
80.                     for(int l = 0; l < N; ++l)
81.                         inv[k][l] -= inv[i][l] * A[k][j];
82.                     for(int l = N; l >= j; --l)
83.                         A[k][l] -= A[i][l] * A[k][j];
84.                 }
85.                 ++i;
86.             }
87.             ++j;
88.         }
89.         return inv;
90.     }
91.
92. Matriz GeneradorMatriz(){
93.     Matriz A = NuevaMatriz();
94.     for(int i = 0; i < N; ++i)
95.         for(int j = 0; j < N; ++j)
96.             A[i][j] = rand() % 10;
97.     return A;
98. }
99.
100. void EscribirMatriz(Matriz A, char* nombre){
101.     FILE * fp = fopen(nombre, "w");
102.     for(int i = 0; i < N; ++i){
103.         for(int j = 0; j < N; ++j)
104.             fprintf(fp, "%0.3lf ", A[i][j]);
105.         fprintf(fp, "\n");
106.     }
107.     fclose(fp);
108. }
109.
110. Matriz LeerMatriz(char* nombre){
111.     Matriz A = NuevaMatriz();
112.     FILE * fp = fopen(nombre, "r");
113.     for(int i = 0; i < N; ++i)
114.         for(int j = 0; j < N; ++j)
115.             fscanf(fp, "%lf", &A[i][j]);
116.     fclose(fp);
117.     return A;
118. }
119.
120. void printMatriz(Matriz A){
121.     for(int i = 0; i < N; ++i){
122.         for(int j = 0; j < N; ++j)
123.             printf("%0.3lf\t", A[i][j]);
124.         printf("\n");
125.     }
126. }

```

## Parte en donde se hace uso de los 6 procesos

Para la creación de procesos por copia exacta de código, primero crearemos dos matrices A y B con entradas aleatorias enteras entre 0-9, Luego lanzamos cinco procesos con las operaciones requeridas entre las matrices y guardamos el resultado en un archivo. Esperamos a que terminen estos procesos, y luego lanzamos un sexto proceso que muestra los resultados de los archivos. Al final mostramos el tiempo total.

### Código(Op6proccopia.c)

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <stdlib.h>
04. #include <time.h>
05. #include <sys/wait.h>
06. #include "EstructuraMatriz.h"
07.
08. int main(){
09.     srand(time(NULL));
10.     clock_t tInicio = clock();
11.     int id_proc;
12.     id_proc = fork();
13.     Matriz A = GeneradorMatriz();
14.     Matriz B = GeneradorMatriz();
15.     EscribirMatriz(A, "MatrizA.txt");
16.     EscribirMatriz(B, "MatrizB.txt");
17.     if(id_proc == 0){
18.         EscribirMatriz(Suma(A, B), "Suma.txt");
19.         exit(0);
20.     }
21.     id_proc = fork();
22.     if(id_proc == 0){
23.         EscribirMatriz(Resta(A, B), "Resta.txt");
24.         exit(0);
25.     }
26.     id_proc = fork();
27.     if(id_proc == 0){
28.         EscribirMatriz(Multiplicacion(A, B), "Multiplicacion.txt");
29.         exit(0);
30.     }
31.     id_proc = fork();
32.     if(id_proc == 0){
33.         EscribirMatriz(Transpuesta(A), "TranspuestaA.txt");
34.         EscribirMatriz(Transpuesta(B), "TranspuestaB.txt");
35.         exit(0);
36.     }
37.     id_proc = fork();
38.     if(id_proc == 0){
39.         EscribirMatriz(Inversa(A), "InversaA.txt");
40.         EscribirMatriz(Inversa(B), "InversaB.txt");
41.         exit(0);
42.     }
43.     for(int i = 0; i < 5; ++i) wait(0);
44.     id_proc = fork();
45.     if(id_proc == 0){
46.         printf("Matriz A:\n");
47.         printMatriz(LeerMatriz("MatrizA.txt"));
48.         printf("\nMatriz B:\n");
49.         printMatriz(LeerMatriz("MatrizB.txt"));
50.         printf("\nSuma de matrices:\n");
51.         printMatriz(LeerMatriz("Suma.txt"));
52.         printf("\nResta de matrices:\n");
53.         printMatriz(LeerMatriz("Resta.txt"));
54.         printf("\nMultiplicacion de matrices:\n");
55.         printMatriz(LeerMatriz("Multiplicacion.txt"));
56.         printf("\nTranspuesta de A:\n");
57.         printMatriz(LeerMatriz("TranspuestaA.txt"));
58.         printf("\nTranspuesta de B:\n");
59.         printMatriz(LeerMatriz("TranspuestaB.txt"));
60.         printf("\nInversa de A:\n");
61.         printMatriz(LeerMatriz("InversaA.txt"));
62.         printf("\nInversa de B:\n");
63.         printMatriz(LeerMatriz("InversaB.txt"));
64.         exit(0);
65.     }
66.     wait(0);
67.     clock_t tFin = clock();
68.     printf("\nTiempo: %0.7fs\n", (double)(tFin - tInicio) / CLOCKS_PER_SEC);
69.     return 0;
70. }
```



Resultado al momento de ejecutar y compilar el programa, al final del programa se nos muestra el tiempo que tardo en finalizar el programa. **Nota: se muestra en segundos.**

```
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/Matriz$ gcc Op6proccopia.c
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/Matriz$ ./a.out
Matriz A:
8.000 6.000 8.000 7.000 3.000 5.000 7.000
9.000 4.000 4.000 6.000 3.000 9.000 7.000
8.000 2.000 4.000 5.000 2.000 4.000 0.000
1.000 4.000 5.000 7.000 6.000 3.000 9.000
6.000 3.000 6.000 4.000 2.000 6.000 3.000
7.000 2.000 1.000 8.000 8.000 5.000 5.000
4.000 4.000 4.000 2.000 8.000 9.000 9.000

Matriz B:
3.000 3.000 9.000 4.000 0.000 6.000 1.000
6.000 0.000 1.000 5.000 5.000 9.000 9.000
7.000 5.000 5.000 6.000 9.000 6.000 5.000
8.000 3.000 2.000 2.000 0.000 6.000 6.000
8.000 7.000 7.000 1.000 3.000 9.000 5.000
3.000 5.000 9.000 1.000 7.000 2.000 6.000
5.000 3.000 8.000 4.000 8.000 3.000 1.000

Suma de matrices:
11.000 9.000 17.000 11.000 3.000 11.000 8.000
15.000 4.000 5.000 11.000 8.000 18.000 16.000
15.000 7.000 9.000 11.000 11.000 10.000 5.000
9.000 7.000 7.000 9.000 6.000 9.000 15.000
14.000 10.000 13.000 5.000 5.000 15.000 8.000
10.000 7.000 10.000 9.000 15.000 7.000 11.000
9.000 7.000 12.000 6.000 16.000 12.000 10.000

Resta de matrices:
5.000 3.000 -1.000 3.000 3.000 -1.000 6.000
3.000 4.000 3.000 1.000 -2.000 0.000 -2.000
1.000 -3.000 -1.000 -1.000 -7.000 -2.000 -5.000
-7.000 1.000 3.000 5.000 6.000 -3.000 3.000
-2.000 -4.000 -1.000 3.000 -1.000 -3.000 -2.000
4.000 -3.000 -8.000 7.000 1.000 3.000 -1.000
-1.000 1.000 -4.000 -2.000 0.000 6.000 8.000

Multiplicacion de matrices:
246.000 152.000 254.000 160.000 202.000 250.000 196.000
213.000 152.000 275.000 132.000 184.000 216.000 177.000
132.000 93.000 154.000 82.000 80.000 146.000 110.000
220.000 133.000 193.000 113.000 176.000 201.000 161.000
159.000 113.000 187.000 103.000 141.000 162.000 136.000
208.000 146.000 227.000 93.000 118.000 211.000 153.000
216.000 166.000 273.000 117.000 215.000 213.000 175.000

Transpuesta de A:
8.000 9.000 8.000 1.000 6.000 7.000 4.000
6.000 4.000 2.000 4.000 3.000 2.000 4.000
8.000 4.000 4.000 5.000 6.000 1.000 4.000
7.000 6.000 5.000 7.000 4.000 8.000 2.000
3.000 3.000 2.000 6.000 2.000 8.000 8.000
5.000 9.000 4.000 3.000 6.000 5.000 9.000
7.000 7.000 0.000 9.000 3.000 5.000 9.000

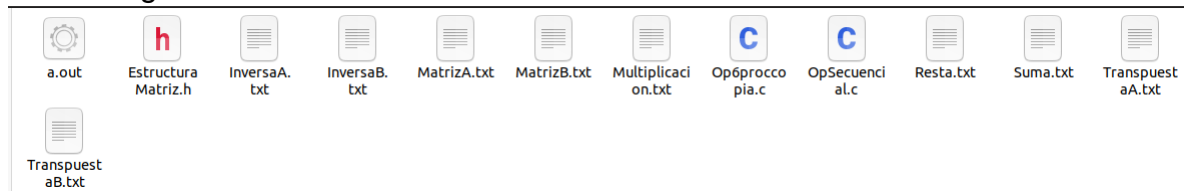
Transpuesta de B:
3.000 6.000 7.000 8.000 8.000 3.000 5.000
3.000 0.000 5.000 3.000 7.000 5.000 3.000
9.000 1.000 5.000 2.000 7.000 9.000 8.000
4.000 5.000 6.000 2.000 1.000 1.000 4.000
0.000 5.000 9.000 0.000 3.000 7.000 8.000
6.000 9.000 6.000 6.000 9.000 2.000 3.000
1.000 9.000 5.000 6.000 5.000 6.000 1.000

Inversa de A:
2.789 -1.759 -7.754 -3.692 5.987 4.341 -1.517
-7.038 4.812 21.544 9.709 -17.207 -11.900 4.369
2.491 -1.804 -7.407 -3.334 6.072 4.052 -1.476
-1.713 1.134 4.665 2.273 -3.573 -2.540 0.779
-0.012 -0.153 0.180 0.017 -0.112 0.019 0.138
-2.403 1.575 6.470 3.012 -4.922 -3.618 1.282
3.575 -2.246 -10.505 -4.725 8.103 5.725 -2.078

Inversa de B:
-0.077 -0.042 -0.088 0.196 -0.028 -0.054 0.183
0.033 -0.139 0.255 -0.036 0.045 0.029 -0.242
0.056 0.011 -0.121 0.028 -0.029 0.057 0.090
0.140 -0.058 0.220 0.064 -0.178 -0.014 -0.133
-0.096 0.052 -0.012 -0.112 0.066 -0.011 0.091
0.008 0.107 -0.051 -0.176 0.168 -0.084 0.001
0.012 0.034 0.007 0.068 -0.084 0.128 -0.109

Tiempo: 0.0005950s
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/Matriz$
```

## Archivos generados:



## Parte en donde no se hace uso de los 6 procesos

Reutilizaremos el código de EstructuraMatriz.h. En este caso simplemente ejecutamos las operaciones de forma secuencial, es decir, pedidas una detrás de otra.

## Código(OpSecuencial.c)

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <stdlib.h>
04. #include <time.h>
05. #include <sys/wait.h>
06. #include "EstructuraMatriz.h"
07.
08. int main(){
09.     srand(time(NULL));
10.     clock_t tInicio = clock();
11.     Matriz A = GeneradorMatriz();
12.     Matriz B = GeneradorMatriz();
13.     EscribirMatriz(A, "MatrizA.txt");
14.     EscribirMatriz(B, "MatrizB.txt");
15.
16.     EscribirMatriz(Suma(A, B), "Suma.txt");
17.     EscribirMatriz(Resta(A, B), "Resta.txt");
18.     EscribirMatriz(Multiplicacion(A, B), "Multiplicacion.txt");
19.     EscribirMatriz(Transpuesta(A), "TranspuestaA.txt");
20.     EscribirMatriz(Transpuesta(B), "TranspuestaB.txt");
21.     EscribirMatriz(Inversa(A), "InversaA.txt");
22.     EscribirMatriz(Inversa(B), "InversaB.txt");
23.
24.     printf("Matriz A:\n");
25.     printMatriz(LeerMatriz("MatrizA.txt"));
26.     printf("\nMatriz B:\n");
27.     printMatriz(LeerMatriz("MatrizB.txt"));
28.     printf("\nSuma de matrices:\n");
29.     printMatriz(LeerMatriz("Suma.txt"));
30.     printf("\nResta de matrices:\n");
31.     printMatriz(LeerMatriz("Resta.txt"));
32.     printf("\nMultiplicacion de matrices:\n");
33.     printMatriz(LeerMatriz("Multiplicacion.txt"));
34.     printf("\nTranspuesta de A:\n");
35.     printMatriz(LeerMatriz("TranspuestaA.txt"));
36.     printf("\nTranspuesta de B:\n");
37.     printMatriz(LeerMatriz("TranspuestaB.txt"));
38.     printf("\nInversa de A:\n");
39.     printMatriz(LeerMatriz("InversaA.txt"));
40.     printf("\nInversa de B:\n");
41.     printMatriz(LeerMatriz("InversaB.txt"));
42.
43.     clock_t tFin = clock();
44.     printf("\nTiempo: %0.7fs\n", (double)(tFin - tInicio) / CLOCKS_PER_SEC);
45.     return 0;
46. }
```

Resultado al momento de ejecutar y compilar el programa, al final del programa se nos muestra el tiempo que tardo en finalizar el programa. **Nota: se muestra en segundos.**



```
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 3/Matriz$ gcc OpSecuencial.c
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 3/Matriz$ ./a.out
```

Matriz A:

1.000	1.000	6.000	7.000	1.000	7.000	6.000
0.000	6.000	5.000	0.000	1.000	4.000	2.000
0.000	8.000	5.000	9.000	9.000	5.000	6.000
4.000	2.000	7.000	7.000	5.000	1.000	7.000
0.000	9.000	4.000	1.000	2.000	2.000	1.000
4.000	2.000	7.000	4.000	8.000	5.000	7.000
0.000	9.000	1.000	2.000	7.000	6.000	1.000

Matriz B:

8.000	3.000	0.000	2.000	7.000	9.000	2.000
3.000	1.000	9.000	5.000	2.000	5.000	7.000
4.000	0.000	8.000	0.000	2.000	7.000	5.000
0.000	2.000	2.000	2.000	3.000	3.000	7.000
0.000	2.000	8.000	9.000	5.000	0.000	3.000
5.000	0.000	5.000	8.000	3.000	6.000	3.000
5.000	4.000	2.000	1.000	4.000	2.000	2.000

Suma de matrices:

9.000	4.000	6.000	9.000	8.000	16.000	8.000
3.000	7.000	14.000	5.000	3.000	9.000	9.000
4.000	8.000	13.000	9.000	11.000	12.000	11.000
4.000	4.000	9.000	9.000	8.000	4.000	14.000
0.000	11.000	12.000	10.000	7.000	2.000	4.000
9.000	2.000	12.000	12.000	11.000	11.000	10.000
5.000	13.000	3.000	3.000	11.000	8.000	3.000

Resta de matrices:

-7.000	-2.000	6.000	5.000	-6.000	-2.000	4.000
-3.000	5.000	-4.000	-5.000	-1.000	-1.000	-5.000
-4.000	8.000	-3.000	9.000	7.000	-2.000	1.000
4.000	0.000	5.000	5.000	2.000	-2.000	0.000
0.000	7.000	-4.000	-8.000	-3.000	2.000	-2.000
-1.000	2.000	2.000	-4.000	5.000	-1.000	4.000
-5.000	5.000	-1.000	1.000	3.000	4.000	-1.000

Multiplicacion de matrices:

100.000	44.000	126.000	92.000	92.000	131.000	124.000
68.000	16.000	126.000	73.000	47.000	93.000	86.000
99.000	68.000	239.000	185.000	137.000	144.000	198.000
106.000	66.000	147.000	92.000	123.000	136.000	138.000
58.000	19.000	143.000	82.000	49.000	90.000	104.000
126.000	66.000	185.000	145.000	141.000	151.000	138.000
66.000	31.000	181.000	161.000	83.000	96.000	123.000

Transpuesta de A:

1.000	0.000	0.000	4.000	0.000	4.000	0.000
1.000	6.000	8.000	2.000	9.000	2.000	9.000
6.000	5.000	5.000	7.000	4.000	7.000	1.000
7.000	0.000	9.000	7.000	1.000	4.000	2.000
1.000	1.000	9.000	5.000	2.000	8.000	7.000
7.000	4.000	5.000	1.000	2.000	5.000	6.000
6.000	2.000	6.000	7.000	1.000	7.000	1.000

Transpuesta de B:

8.000	3.000	4.000	0.000	0.000	5.000	5.000
3.000	1.000	0.000	2.000	2.000	0.000	4.000
0.000	9.000	8.000	2.000	8.000	5.000	2.000
2.000	5.000	0.000	2.000	9.000	8.000	1.000
7.000	2.000	2.000	3.000	5.000	3.000	4.000
9.000	5.000	7.000	3.000	0.000	6.000	2.000
2.000	7.000	5.000	7.000	3.000	3.000	2.000

Inversa de A:

0.151	-0.427	-0.344	0.150	0.312	0.062	0.214
-0.494	1.699	-0.059	1.221	-1.719	-1.098	0.777
1.158	-4.040	0.104	-2.848	4.408	2.559	-1.871
0.688	-2.227	0.016	-1.302	2.233	1.130	-0.801
0.360	-1.605	0.123	-1.168	1.646	1.078	-0.705
0.246	-0.467	-0.080	-0.351	0.440	0.289	-0.072
-2.083	7.238	0.017	4.743	-7.566	-4.223	2.841

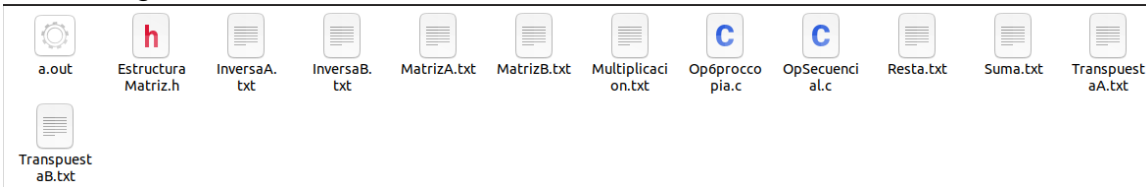
Inversa de B:

-0.702	-1.217	0.659	0.431	-0.102	0.859	0.666
1.211	2.493	-1.411	-0.951	-0.015	-1.506	-0.798
0.076	0.211	-0.011	-0.159	0.065	-0.176	-0.063
0.162	0.382	-0.295	-0.131	-0.001	-0.101	-0.151
-0.575	-1.484	0.826	0.565	0.151	0.705	0.445
0.862	1.519	-0.799	-0.589	-0.004	-0.922	-0.729
-0.537	-0.897	0.479	0.508	-0.077	0.602	0.411

Tiempo: 0.0008040s

```
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 3/Matriz$
```

## Archivos generados:



Vemos que la aplicación secuencial tardó un poco más en ejecutarse que la que usa copia exacta de código, con base en el anterior resultado podemos deducir que cada proceso se tardó un poco más en ser ejecutado en el programa secuencial y podemos concluir que usar copia exacta de código es mejor en cuanto a tiempo de ejecución, es decir, para programas en donde necesitamos bajar el tiempo, usar copia exacta de código es una muy buena opción.

### 2.1.6. Ejemplo de creación de procesos por sustitución de código

Por medio del siguiente código podemos observar el uso de la llamada al sistema **fork()**, **execv(const char\* path, char\* const argv[])** y **wait(int \*status)**. La llamada al sistema **execv(const char\* path, char\* const argv[])** hace la creación de un proceso por sustitución de código, el cual es "hola.c".

#### Código(Sustitucion.c)

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <sys/types.h>
04. #include <sys/wait.h>
05. #include <stdlib.h>
06.
07. int main(){
08.     pid_t pid;
09.     char *argv[3];
10.     argv[0]="/home/edmundojm/Escritorio/Sistemas Operativos/Practica 3/hola"; /*Ruta*/
11.     argv[1]="Desde el Hijo";
12.     argv[2]=NULL;
13.     if((pid=fork())== -1)
14.         printf("Error al crear el proceso hijo\n");
15.     if(pid==0){
16.         printf("Soy el hijo ejecutando:%s\n",argv[0]);
17.         execv(argv[0],argv);
18.     }else{
19.         wait(0);
20.         printf("Soy el Padre\n");
21.         exit(0);
22.     }
23. }
```

#### Código(hola.c)

```
01. /* hola.c Programa que sera invocado*/
02.
03. #include <stdio.h>
04. #include <string.h>
05. #include <stdlib.h>
06.
07. int main(int argc, char *argv[]){
08.     char mensaje[100];
09.     strcpy(mensaje,"Hola Mundo ");
10.     strcat(mensaje,argv[1]);
11.     printf("%s\n",mensaje);
12.     exit(0);
13. }
```

Resultado al momento de ejecutar y compilar el programa:

```
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 3$ gcc hola.c -o hola
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 3$ gcc Sustitucion.c
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 3$ ./a.out
Soy el hijo ejecutando:/home/edmundojism/Escritorio/Sistemas Operativos/Practica 3/hola
Hola Mundo Desde el Hijo
Soy el Padre
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 3$
```

Con el código realizamos dos procesos con la llamada al sistema **fork()** y podemos observar con la llamada al sistema **wait(int \*status)** como configura al sistema operativo para que se ejecute primero el proceso hijo, mientras que el proceso padre espera a que finalice su proceso hijo.

En el proceso hijo observamos el uso de la llamada al sistema **execv(const char\* path, char\* const argv[])** que hace la creación de un proceso por sustitución de código que imprime el mensaje “Hola Mundo Desde el Hijo”, indicando que es el hijo quien lo ejecuto y terminando el proceso hijo se regresa al proceso padre que imprime el mensaje “Soy el padre”.

Pero que pasara si quitáramos wait();?

```
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 3$ ./a.out
Soy el Padre
edmundojism@edmundojism-VB:~/Escritorio/Sistemas Operativos/Practica 3$ Soy el hijo ejecutando:/home/edmundojism/Escritorio/Sistemas Operativos/Practica
3/hola
Hola Mundo Desde el Hijo
```

En caso de que no tuviera la llamada al sistema **wait (int \*status)** se ejecutaría primero el proceso padre y después el proceso hijo en la mayoría de los casos.

Por último, y si quisiera que tanto el hijo como padre ejecutaran hola.c?

Sencillo, solo tendríamos que copiar la llamada al sistema **execv(const char\* path, char\* const argv[])** y cambiar el valor de argv[1] por “Desde el Padre”, por lo que el código quedaría:

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <sys/types.h>
04. #include <sys/wait.h>
05. #include <stdlib.h>
06.
07. int main(){
08.     pid_t pid;
09.     char *argv[3];
10.     argv[0]="/home/edmundojism/Escritorio/Sistemas Operativos/Practica 3/hola"; /*Ruta*/
11.     argv[1]="Desde el Hijo";
12.     argv[2]=NULL;
13.     if((pid=fork())== -1)
14.         printf("Error al crear el proceso hijo\n");
15.     if(pid==0){
16.         printf("Soy el hijo ejecutando:%s\n",argv[0]);
17.         execv(argv[0],argv);
18.     }else{
19.         wait(0);
20.         printf("Soy el Padre\n");
21.         argv[1]="Desde el Padre";
22.         printf("Soy el padre ejecutando:%s\n",argv[0]);
23.         execv(argv[0],argv);
24.         exit(0);
25.     }
26. }
```

El resultado sería el siguiente:

```
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$ gcc Sustitucion.c
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$ ./a.out
Soy el hijo ejecutando:/home/edmundojm/Escritorio/Sistemas Operativos/Practica 3/hola
Hola Mundo Desde el Hijo
Soy el Padre
Soy el padre ejecutando:/home/edmundojm/Escritorio/Sistemas Operativos/Practica 3/hola
Hola Mundo Desde el Padre
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3$
```

### 2.1.7. Tres procesos usando sustitución de código

Nuestro código principal ser 3CodigoSustitucion.c el cual en su código fuente hace una sustitución de código con los siguientes 3 programas:

- Resolver ecuación algebraica de segundo grado mediante la fórmula general. (SutituirEc.c)
- Mostrar serie Fibonacci para un número N (SustituirFibo.c)
- Multiplicación de dos matrices de 7x7 de tipo entero(SustituirMat.c)

#### Código(3CodigoSustitucion.c)

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <sys/types.h>
04. #include <sys/wait.h>
05. #include <stdlib.h>
06. #include <string.h>
07.
08. int main(){
09.     int id_proc;
10.     pid_t pid,pid2;
11.     char *argv[3];
12.     argv[0]="/home/edmundojm/Escritorio/Sistemas Operativos/Practica 3/3Procesos/Sustituir1";
13.     argv[1]=NULL;
14.     if((pid=fork())==0){
15.         if((pid2=fork())==0){
16.             printf("Aquí es donde entramos nosotros (los nietos de Odin)\n");
17.             for(int i=0;i<=2;++i){
18.                 id_proc = fork();
19.                 if(id_proc==0){
20.                     if(i==1)
21.                         argv[0]="/home/edmundojm/Escritorio/Sistemas Operativos/Practica 3/3Procesos/Sustituir2";
22.                     if(i==2)
23.                         argv[0]="/home/edmundojm/Escritorio/Sistemas Operativos/Practica 3/3Procesos/Sustituir3";
24.                     execv(argv[0],argv);
25.                     exit(0);
26.                 }
27.             }
28.         }else{
29.             printf("Soy el hijo de Odin\n");
30.         }
31.     }else{
32.         printf("Soy Odin el padre de todos\n");
33.         exit(0);
34.     }
35. }
```

#### Código(SutituirEc.c) compilado como Sustituir1:

```
01. #include <stdio.h>
02. #include <math.h>
03.
04. int main(){
05.     int a,b,c,dentroR;
06.     float s1,s2;
07.     a=-4;
08.     b=5;
09.     c=0;
10.     dentroR = b*b-4*a*c;
11.     if (dentroR >=0) {
12.         s1 = (-b+sqrt(dentroR))/(2*a);
13.         s2 = (-b-sqrt(dentroR))/(2*a);
14.         printf("x1 = %.2f\n",s1);
15.         printf("x2 = %.2f\n",s2);
16.     }else{
17.         dentroR*=-1;
18.         float si;
19.         s1 = (-b)/(2*a);
20.         si = sqrt(dentroR)/(2*a);
21.         printf("x1 = %.2f + %.2f i\n",s1,si);
22.         printf("x2 = %.2f - %.2f i\n",s1,si);
23.     }
24.     return 0;
25. }
```

Código(SustituirFibo.c) compilado como Sustituir2:

```
01. #include <stdio.h>
02.
03. int main()
04. {
05.     int i = 0;
06.     int j = 1;
07.     int n=7;
08.     for (int k = 1; k < n; k++){
09.         int t;
10.         t = i + j;
11.         i = j;
12.         j = t;
13.     }
14.     printf("Fibonacci de %d es: %d\n",n,j);
15. }
```

Código(EstucuraMatriz.c) primer imagen y Código(SustituirMat.c) segunda imagen, el ultimo es compilado como Sustituir3:

```
01. #include <stdbool.h>
02. #include <math.h>
03. typedef double* Vector;
04. typedef Vector* Matriz;
05. #define N 7
06.
07. Matriz NuevaMatriz(){
08.     int M = N;
09.     if(N&1)
10.         M+=1;
11.     Matriz A = calloc(M, sizeof(Vector));
12.     for(int i = 0; i < N; ++i)
13.         A[i] = calloc(M, sizeof(double));
14.     return A;
15. }
16.
17. bool esCero(double x){
18.     return fabs(x) < 1e-8;
19. }
20.
21. Matriz Multiplicacion(Matriz A, Matriz B){
22.     Matriz C = NuevaMatriz();
23.     for(int i = 0; i < N; ++i)
24.         for(int j = 0; j < N; ++j)
25.             for(int k = 0; k < N; ++k)
26.                 C[i][j] += A[i][k] * B[k][j];
27.     return C;
28. }
29.
30. Matriz GeneradorMatriz(){
31.     Matriz A = NuevaMatriz();
32.     for(int i = 0; i < N; ++i)
33.         for(int j = 0; j < N; ++j)
34.             A[i][j] = rand() % 10;
35.     return A;
36. }
37.
38. void printMatriz(Matriz A){
39.     for(int i = 0; i < N; ++i){
40.         for(int j = 0; j < N; ++j)
41.             printf("%.3lf\t", A[i][j]);
42.         printf("\n");
43.     }
44. }
```

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <stdlib.h>
04. #include <time.h>
05. #include <sys/wait.h>
06. #include "EstructuraMatriz.h"
07.
08. int main(){
09.     srand(time(NULL));
10.     Matriz A = GeneradorMatriz();
11.     Matriz B = GeneradorMatriz();
12.     printf("Matriz A:\n");
13.     printMatriz(A);
14.     printf("\nMatriz B:\n");
15.     printMatriz(B);
16.     printf("\nMultiplicacion de matrices:\n");
17.     printMatriz(Multiplicacion(A, B));
18.     return 0;
19. }
```

## Compilación y ejecución del código:

```
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/3Procesos$ gcc SustituirEc.c -o Sustituir1 -lm
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/3Procesos$ gcc SustituirFibo.c -o Sustituir2
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/3Procesos$ gcc SustituirMat.c -o Sustituir3
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/3Procesos$ gcc 3CodigoSustitucion.c
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/3Procesos$ ./a.out
Soy Odin el padre de todos
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/3Procesos$ Soy el hijo de Odin
Aqui es donde entramos nosotros (los nietos de Odin)
Matriz A:
7.000 1.000 4.000 5.000 8.000 7.000 4.000
6.000 8.000 6.000 9.000 7.000 9.000 4.000
2.000 2.000 2.000 9.000 6.000 5.000 6.000
1.000 9.000 4.000 5.000 2.000 9.000 0.000
1.000 2.000 2.000 9.000 4.000 7.000 4.000
4.000 6.000 0.000 3.000 6.000 8.000 2.000
3.000 8.000 8.000 7.000 2.000 1.000 7.000

Matriz B:
9.000 8.000 5.000 2.000 7.000 9.000 9.000
1.000 9.000 1.000 3.000 1.000 4.000 4.000
7.000 1.000 1.000 8.000 4.000 7.000 0.000 7.000
Fibonacci de 7 es: 13
3.000 8.000 1.000 9.000 6.000 9.000 6.000
1.000 2.000 3.000 0.000 3.000 9.000 4.000
0.000 8.000 5.000 4.000 9.000 7.000 9.000
3.000 3.000 5.000 2.000 4.000 5.000 6.000

Multiplicacion de matrices:
127.000 193.000 152.000 114.000 211.000 253.000 244.000
150.000 296.000 181.000 185.000 264.000 313.000 315.000
85.000 178.000 110.000 131.000 171.000 226.000 199.000
63.000 209.000 102.000 126.000 161.000 171.000 192.000
68.000 176.000 99.000 133.000 168.000 203.000 188.000
63.000 192.000 97.000 89.000 150.000 207.000 186.000
135.000 193.000 140.000 143.000 170.000 182.000 216.000
x1 = -0.00
x2 = 1.25
```

¿Es posible un funcionamiento 100 % concurrente de su aplicación? Explique porque sí o no es 100% concurrente su aplicación.

No, no es posible 100 % concurrente, y la razón reside en la creación de los 3 procesos, ya que cuando estos se ejecutan hay un momento en el que la computadora se confunde y ejecuta dos procesos al mismo tiempo como se muestra en las capturas de pantalla y viceversa.

### 2.1.7. Programa con 6 procesos por sustitución de código (Matrices)

Reutilizaremos el código de EstructuraMatriz.h. Haremos lo mismo que en punto 5, pero ahora creando un proceso por sustitución por cada operación y otro para que muestre los resultados. Para lograrlo haremos otro programa llamado ElegirProcesoM.out que recibirá un parámetro del 1 al 6 indicando la acción a realizar. Las matrices ya deben de existir en los archivos A.txt y B.txt, pues este programa las leerá de ahí.

Finalmente, desde el programa principal crearemos 6 procesos por copia exacta y dentro de cada uno crearemos el proceso correspondiente por sustitución.

Código(ElegirProcesoM.c) compilado como ElegirProcesoM.out:

```
01. #include <stdio.h>
02. #include <stdlib.h>
03. #include "EstructuraMatriz.h"
04.
05. int main(int argc, char *argv[]){
06.     int op;
07.     sscanf(argv[1], "%d", &op);
08.     Matriz A = LeerMatriz("A.txt");
09.     Matriz B = LeerMatriz("B.txt");
10.     if(op == 1)
11.         EscribirMatriz(Suma(A,B), "Suma.txt");
12.     else if(op == 2)
13.         EscribirMatriz(Resta(A,B), "Resta.txt");
14.     else if(op == 3)
15.         EscribirMatriz(Multiplicacion(A,B), "Multiplicacion.txt");
16.     else if(op == 4){
17.         EscribirMatriz(Transpuesta(A), "TranspuestaA.txt");
18.         EscribirMatriz(Transpuesta(B), "TranspuestaB.txt");
19.     }else if(op == 5){
20.         EscribirMatriz(Inversa(A), "InversaA.txt");
21.         EscribirMatriz(Inversa(B), "InversaB.txt");
22.     }else if(op == 6){
23.         printf("Matriz A:\n");
24.         printMatriz(LeerMatriz("A.txt"));
25.         printf("\nMatriz B:\n");
26.         printMatriz(LeerMatriz("B.txt"));
27.         printf("\nSuma de matrices:\n");
28.         printMatriz(LeerMatriz("Suma.txt"));
29.         printf("\nResta de matrices:\n");
30.         printMatriz(LeerMatriz("Resta.txt"));
31.         printf("\nMultiplicacion de matrices:\n");
32.         printMatriz(LeerMatriz("Multiplicacion.txt"));
33.         printf("\nTranspuesta de A:\n");
34.         printMatriz(LeerMatriz("TranspuestaA.txt"));
35.         printf("\nTranspuesta de B:\n");
36.         printMatriz(LeerMatriz("TranspuestaB.txt"));
37.         printf("\nInversa de A:\n");
38.         printMatriz(LeerMatriz("InversaA.txt"));
39.         printf("\nInversa de B:\n");
40.         printMatriz(LeerMatriz("InversaB.txt"));
41.     }
42.     return 0;
43. }
```

Código(Matriz6ProcSust.c)

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <stdlib.h>
04. #include <time.h>
05. #include <sys/wait.h>
06. #include <sys/types.h>
07. #include "EstructuraMatriz.h"
08.
09. void ejecutar(int op){
10.     char *argv[3];
11.     argv[0] = "/home/edmundojm/Escritorio/Sistemas Operativos/Practica 3/Matriz6ProcesosSustitucion/ElegirProcesoM.out";
12.     char str[2];
13.     sprintf(str, "%d", op);
14.     argv[1] = str;
15.     argv[2] = NULL;
16.     execv(argv[0], argv);
17. }
18.
19. int main(){
20.     srand(time(NULL));
21.     clock_t tInicio = clock();
22.     Matriz A = GeneradorMatriz();
23.     Matriz B = GeneradorMatriz();
24.     EscribirMatriz(A, "A.txt");
25.     EscribirMatriz(B, "B.txt");
26.     int id_proc;
27.     for(int i = 1; i <= 5; ++i){
28.         id_proc = fork();
29.         if(id_proc == 0){
30.             ejecutar(i);
31.             exit(0);
32.         }
33.     }
34.     for(int i = 1; i <= 5; ++i)
35.         wait(0);
36.     id_proc = fork();
37.     if(id_proc == 0){
38.         ejecutar(6);
39.         exit(0);
40.     }
41.     wait(0);
42.     clock_t tFin = clock();
43.     printf("\nTiempo: %.7fs\n", (double)(tFin - tInicio) / CLOCKS_PER_SEC);
44.     return 0;
45. }
```



## Compilación y ejecución del código:

```
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/Matriz6ProcesosSustitucion$ gcc ElegirProcesoM.c -o ElegirProcesoM.out
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/Matriz6ProcesosSustitucion$ gcc Matriz6ProcSust.c
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/Matriz6ProcesosSustitucion$ ./a.out

Matriz A:
3.000 1.000 9.000 6.000 2.000 5.000 6.000
4.000 5.000 3.000 8.000 7.000 1.000 1.000
4.000 4.000 6.000 2.000 4.000 0.000 5.000
1.000 4.000 8.000 8.000 1.000 8.000 6.000
3.000 8.000 5.000 9.000 1.000 6.000 5.000
4.000 4.000 1.000 0.000 1.000 7.000 9.000
8.000 0.000 2.000 4.000 4.000 8.000 8.000

Matriz B:
8.000 0.000 3.000 1.000 7.000 3.000 9.000
8.000 2.000 8.000 4.000 2.000 5.000 3.000
3.000 2.000 0.000 9.000 8.000 2.000 2.000
9.000 9.000 1.000 9.000 9.000 5.000 5.000
6.000 3.000 4.000 6.000 3.000 9.000 8.000
0.000 3.000 9.000 1.000 7.000 7.000 7.000
1.000 3.000 0.000 6.000 7.000 2.000 6.000

Suma de matrices:
11.000 1.000 12.000 7.000 9.000 8.000 15.000
12.000 7.000 11.000 12.000 9.000 6.000 4.000
7.000 6.000 6.000 11.000 12.000 2.000 7.000
10.000 13.000 9.000 17.000 10.000 13.000 11.000
9.000 11.000 9.000 15.000 4.000 15.000 13.000
4.000 7.000 10.000 1.000 8.000 14.000 16.000
9.000 3.000 2.000 10.000 11.000 10.000 14.000

Resta de matrices:
-5.000 1.000 6.000 5.000 -5.000 2.000 -3.000
-4.000 3.000 -5.000 4.000 5.000 -4.000 -2.000
1.000 2.000 6.000 -7.000 -4.000 -2.000 3.000
-8.000 -5.000 7.000 -1.000 -8.000 3.000 1.000
-3.000 5.000 1.000 3.000 -2.000 -3.000 -3.000
4.000 1.000 -8.000 -1.000 -6.000 0.000 2.000
7.000 -3.000 2.000 -2.000 -3.000 6.000 2.000

Multiplicacion de matrices:
131.000 113.000 76.000 195.000 232.000 127.000 165.000
196.000 115.000 97.000 172.000 169.000 155.000 166.000
129.000 65.000 62.000 146.000 149.000 100.000 132.000
148.000 141.000 119.000 211.000 252.000 156.000 177.000
195.000 143.000 140.000 203.000 238.000 165.000 186.000
82.000 61.000 111.000 96.000 159.000 110.000 161.000
138.000 100.000 116.000 142.000 232.000 156.000 232.000

Transpuesta de A:
3.000 4.000 4.000 1.000 3.000 4.000 8.000
1.000 5.000 4.000 4.000 8.000 4.000 0.000
9.000 3.000 6.000 8.000 5.000 1.000 2.000
6.000 8.000 2.000 8.000 9.000 0.000 4.000
2.000 7.000 4.000 1.000 1.000 1.000 4.000
5.000 1.000 0.000 8.000 6.000 7.000 8.000
6.000 1.000 5.000 6.000 5.000 9.000 8.000

Transpuesta de B:
8.000 8.000 3.000 9.000 6.000 0.000 1.000
0.000 2.000 2.000 9.000 3.000 3.000 3.000
3.000 8.000 0.000 1.000 4.000 9.000 0.000
1.000 4.000 9.000 9.000 6.000 1.000 6.000
7.000 2.000 8.000 9.000 3.000 7.000 7.000
3.000 5.000 2.000 5.000 9.000 7.000 2.000
9.000 3.000 2.000 5.000 8.000 7.000 6.000

Inversa de A:
-0.827 -0.430 0.632 0.483 0.125 -0.621 0.537
-0.702 -0.223 0.465 0.490 0.028 -0.307 0.224
-0.798 -0.361 0.639 0.701 -0.077 -0.536 0.369
1.236 0.453 -0.849 -0.979 0.171 0.610 -0.513
-0.167 0.154 0.060 0.233 -0.244 0.045 -0.004
-1.328 -0.434 0.755 1.129 -0.166 -0.678 0.597
1.820 0.651 -1.153 -1.414 0.096 1.105 -0.844

Inversa de B:
0.068 0.023 0.012 0.039 -0.004 -0.045 -0.092
-0.074 0.028 -0.162 0.118 -0.053 0.011 0.111
-0.043 0.162 -0.049 -0.046 -0.086 0.025 0.123
-0.075 0.111 0.021 -0.067 0.014 -0.072 0.172
0.071 -0.108 0.139 0.036 -0.045 0.089 -0.172
0.012 -0.188 0.143 0.033 0.166 0.085 -0.321
0.014 0.061 -0.153 -0.051 0.011 -0.057 0.262

Tiempo: 0.0004020s
edmundojm@edmundojm-VB:~/Escritorio/Sistemas Operativos/Practica 3/Matriz6ProcesosSustitucion$
```



Vemos que el tiempo ahora fue de 0.0004s, superando a la aplicación de copia exacta, lo que quiere decir que lo mas conveniente sería ejecutar una aplicación mediante sustitución de código si es que queremos velocidad.

## 2.2. Windows

### 2.2.1. Ejemplo de creación de procesos por sustitución de código

Código(EjSustitucionCodigoW.c)

```
01. #include <windows.h>
02. #include <stdio.h>
03.
04. int main(int argc, char *argv[]){
05.     STARTUPINFO si; /* Estructura de información inicial para Windows */
06.     PROCESS_INFORMATION pi; /* Estructura de información del adm. de procesos */
07.     int i;
08.     ZeroMemory(&si, sizeof(si));
09.     si.cb = sizeof(si);
10.     ZeroMemory(&pi, sizeof(pi));
11.     if (argc != 2){
12.         printf("Usar:%s Nombre programa hijo\n", argv[0]);
13.         return;
14.     }
15.     if (!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0,
16.         NULL, NULL, &si, &pi)) {
17.         printf("Fallo al invocar CreateProcess(%d)\n", GetLastError());
18.         return;
19.     }
20.     printf("Soy el proceso padre\n");
21.     WaitForSingleObject(pi.hProcess, INFINITE);
22.     CloseHandle(pi.hProcess);
23.     CloseHandle(pi.hThread);
24. }
```

Código(Hijo.c)

```
01. #include <windows.h>
02. #include <stdio.h>
03.
04. int main(){
05.     printf("Soy el proceso hijo\n");
06.     exit(0);
07. }
```

Compilación y ejecución del programa EjSustitucionCodigoW pasando como parámetro el ejecutable del programa Hijo. Notar que se nos muestra advertencias de no tener valor de return, pero esto no afecta a la funcionalidad correcta del programa.

```

C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>gcc Hijo.c -o Hijo.exe

C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>gcc EjSustitucionCodigoW.c -o Ejem.exe
EjSustitucionCodigoW.c: In function 'main':
EjSustitucionCodigoW.c:13:9: warning: 'return' with no value, in function returning non-void
    return;
    ^~~~~~
EjSustitucionCodigoW.c:4:5: note: declared here
int main(int argc, char *argv[]){
    ^~~~~
EjSustitucionCodigoW.c:18:9: warning: 'return' with no value, in function returning non-void
    return;
    ^~~~~~
EjSustitucionCodigoW.c:4:5: note: declared here
int main(int argc, char *argv[]){
    ^~~~~

C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>Ejem.exe Hijo.exe
Soy el proceso padre
Soy el proceso hijo

C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>

```

Ahora experimentemos con el programa, trataremos de quitarle los warnings que nos aparece dándole un valor los return de 0 y agregando un return 0 al final del código. Ahora veamos la compilación y ejecución del programa con ese cambio.

```

C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>gcc Hijo.c -o Hijo.exe

C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>gcc EjSustitucionCodigoW.c -o Ejem.exe

C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>Ejem.exe Hijo.exe
Soy el proceso padre
Soy el proceso hijo

C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>

```

Vemos que no cambia en el funcionamiento, de hecho, logramos quitar las advertencias con lo que se mencionó anteriormente, así concluimos la experimentación de este código.

Ahora comparando con Linux vemos que la diferencia de un programa por sustitución de código en Linux con respecto a Windows se observa que en el programa principal necesita pasar como argumento el programa que ejecutara el proceso secundario (Hijo.exe), el cual a su vez es el argumento de la llamada al sistema. En este caso, si no le pasamos ningún argumento al programa Ejem.exe, nos mostrara un mensaje de la línea 12 en donde argv[0]="Ejem.exe".

### 2.2.2. Creación de árbol de procesos en Windows

El siguiente programa recibe como argumento el nivel del árbol en el que estamos, para posteriormente revisar cuántos hijos deberá crear el programa, cada uno con el nivel incrementado en uno. También tenemos que guardar la información de cada proceso para esperar a que termine.

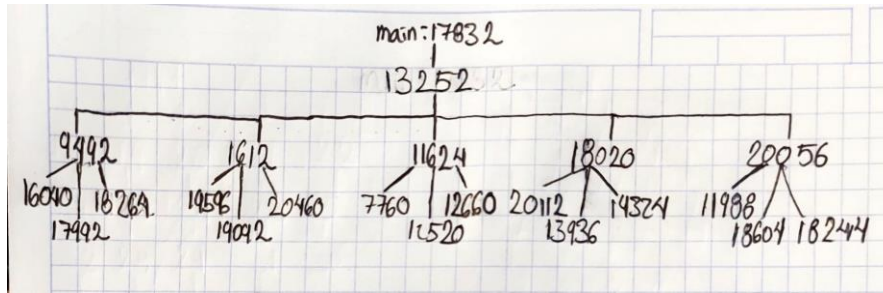
## Código(ArbolProcesosW.c)

```
01. #include <stdio.h>
02. #include <windows.h>
03.
04. char* spaces(int n){
05.     char *s = calloc(n + 1, sizeof(char));
06.     memset(s, '.', n);
07.     s[n] = '\\0';
08.     return s;
09. }
10.
11. int main(int argc, char *argv[]){
12.     int nivel, hijos;
13.     if(argc < 2)
14.         nivel = 0;
15.     else
16.         sscanf(argv[1], "%d", &nivel);
17.     if(nivel == 0)
18.         printf("Main: %d\\n", GetCurrentProcessId());
19.     else{
20.         printf("%sInicio proceso hijo %d\\n", spaces(nivel), GetCurrentProcessId());
21.     }
22.     if(nivel <= 2){
23.         char hijosPorNivel[3] = {1, 5, 3};
24.         hijos = hijosPorNivel[nivel];
25.         HANDLE *process, *threads;
26.         DWORD *pids;
27.         process = calloc(hijos, sizeof(HANDLE));
28.         threads = calloc(hijos, sizeof(HANDLE));
29.         pids = calloc(hijos, sizeof(DWORD));
30.         for(int i = 0; i < hijos; ++i){
31.             STARTUPINFO si;
32.             PROCESS_INFORMATION pi;
33.             ZeroMemory(&si, sizeof(si));
34.             si.cb = sizeof(si);
35.             ZeroMemory(&pi, sizeof(pi));
36.             char args[100];
37.             sprintf(args, "%s %d", argv[0], nivel + 1);
38.             CreateProcess(NULL, args, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);
39.             process[i] = pi.hProcess;
40.             threads[i] = pi.hThread;
41.             pids[i] = pi.dwProcessId;
42.             printf("%sCreando proceso hijo %d, padre: %d\\n", spaces(nivel), pids[i], GetCurrentProcessId());
43.         }
44.         for(int i = 0; i < hijos; ++i){
45.             WaitForSingleObject(process[i], INFINITE);
46.             CloseHandle(process[i]);
47.             CloseHandle(threads[i]);
48.             printf("%sFin proceso hijo %d\\n", spaces(nivel + 1), pids[i]);
49.         }
50.     }
51.     return 0;
52. }
```

Resultado al momento de ejecutar y compilar el programa:

```
C:\Windows\system32\cmd.exe
C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>gcc ArbolProcesosW.c -o ArbolW.exe
C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>ArbolW.exe
Main: 17832
Creando proceso hijo 13252, padre: 17832
..Inicio proceso hijo 13252
..Creando proceso hijo 9492, padre: 13252
..Creando proceso hijo 1612, padre: 13252
..Creando proceso hijo 11624, padre: 13252
..Inicio proceso hijo 1612
..Inicio proceso hijo 9492
..Creando proceso hijo 18020, padre: 13252
..Creando proceso hijo 20056, padre: 13252
..Creando proceso hijo 16040, padre: 9492
..Inicio proceso hijo 11624
..Creando proceso hijo 19596, padre: 1612
..Creando proceso hijo 17992, padre: 9492
..Inicio proceso hijo 20056
...Inicio proceso hijo 19596
..Creando proceso hijo 19092, padre: 1612
...Inicio proceso hijo 16040
..Creando proceso hijo 18264, padre: 9492
..Creando proceso hijo 20460, padre: 1612
...Fin proceso hijo 19596
..Fin proceso hijo 16040
..Inicio proceso hijo 18020
..Creando proceso hijo 11988, padre: 20056
...Inicio proceso hijo 19092
..Creando proceso hijo 18604, padre: 20056
..Creando proceso hijo 20112, padre: 18020
...Inicio proceso hijo 20460
..Creando proceso hijo 18244, padre: 20056
..Creando proceso hijo 13936, padre: 18020
..Creando proceso hijo 7760, padre: 11624
...Inicio proceso hijo 18264
...Inicio proceso hijo 11988
...Fin proceso hijo 19092
...Fin proceso hijo 20460
...Inicio proceso hijo 18604
...Inicio proceso hijo 7760
..Creando proceso hijo 14324, padre: 18020
..Creando proceso hijo 12520, padre: 11624
...Inicio proceso hijo 20112
...Inicio proceso hijo 13936
..Creando proceso hijo 12660, padre: 11624
...Fin proceso hijo 7760
...Fin proceso hijo 11988
...Inicio proceso hijo 18244
...Inicio proceso hijo 14324
...Fin proceso hijo 20112
...Inicio proceso hijo 12520
...Fin proceso hijo 18604
...Fin proceso hijo 13936
...Fin proceso hijo 18244
...Fin proceso hijo 14324
...Fin proceso hijo 12520
...Inicio proceso hijo 12660
...Fin proceso hijo 12660
...Inicio proceso hijo 17992
...Fin proceso hijo 17992
...Fin proceso hijo 18264
..Fin proceso hijo 9492
..Fin proceso hijo 1612
..Fin proceso hijo 11624
..Fin proceso hijo 18020
..Fin proceso hijo 20056
..Fin proceso hijo 13252
C:\Users\Edmundo J Sanchez M\Desktop\S0\P3>
```

El árbol de procesos creado por el programa de acuerdo con los PID's de cada uno es el siguiente:



El cual coincide con el árbol solicitado.

### 2.2.3. Programa con 6 procesos por sustitución de código (Matrices)

Usaremos de nuevo el código de EstructuraMatriz.h y ElegirProcesoM.c (ahora compilado como ElegirProcesoM.exe) del punto 8 de la sección de Linux, pues la funcionalidad lógica de la aplicación que queremos no cambia, además de usar OpSecuencial.c para ejecutar de forma secuencial el programa. Adaptaremos el código de la creación de los procesos en Windows, dado todo lo anterior solo se anexará el código en donde se ejecutarán los 6 procesos y el secuencial.

Código(Matriz6ProcSust.c)

```

01. #include <stdio.h>
02. #include <windows.h>
03. #include <time.h>
04. #include "EstructuraMatriz.h"
05.
06. PROCESS_INFORMATION ejecutar(int op){
07.     STARTUPINFO si;
08.     PROCESS_INFORMATION pi;
09.     ZeroMemory(&si, sizeof(si));
10.     si.cb = sizeof(si);
11.     ZeroMemory(&pi, sizeof(pi));
12.     char str[100];
13.     sprintf(str, "ElegirProcesoM.exe %d", op);
14.     CreateProcess("ElegirProcesoM.exe", str, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);
15.     return pi;
16. }
17.
18. int main(){
19.     srand(time(NULL));
20.     clock_t tInicio = clock();
21.     Matriz A = GeneradorMatriz();
22.     Matriz B = GeneradorMatriz();
23.     EscribirMatriz(A, "A.txt");
24.     EscribirMatriz(B, "B.txt");
25.     HANDLE process[5], thread[5];
26.     for(int i = 0; i < 5; ++i){
27.         PROCESS_INFORMATION pi = ejecutar(i + 1);
28.         process[i] = pi.hProcess;
29.         thread[i] = pi.hThread;
30.     }
31.     for(int i = 0; i < 5; ++i){
32.         WaitForSingleObject(process[i], INFINITE);
33.         CloseHandle(process[i]);
34.         CloseHandle(thread[i]);
35.     }
36.     PROCESS_INFORMATION pi = ejecutar(6);
37.     WaitForSingleObject(pi.hProcess, INFINITE);
38.     CloseHandle(pi.hProcess);
39.     CloseHandle(pi.hThread);
40.     clock_t tFin = clock();
41.     printf("\nTiempo: %.7fs\n", (double)(tFin - tInicio) / CLOCKS_PER_SEC);
42.     return 0;
43. }
  
```

## Código(OpSecuencial.c)

```
01. #include <stdio.h>
02. #include <unistd.h>
03. #include <stdlib.h>
04. #include <time.h>
05. #include "EstructuraMatriz.h"
06.
07. int main(){
08.     srand(time(NULL));
09.     clock_t tInicio = clock();
10.     Matriz A = GeneradorMatriz();
11.     Matriz B = GeneradorMatriz();
12.     EscribirMatriz(A, "MatrizA.txt");
13.     EscribirMatriz(B, "MatrizB.txt");
14.
15.     EscribirMatriz(Suma(A, B), "Suma.txt");
16.     EscribirMatriz(Resta(A, B), "Resta.txt");
17.     EscribirMatriz(Multiplicacion(A, B), "Multiplicacion.txt");
18.     EscribirMatriz(Transpuesta(A), "TranspuestaA.txt");
19.     EscribirMatriz(Transpuesta(B), "TranspuestaB.txt");
20.     EscribirMatriz(Inversa(A), "InversaA.txt");
21.     EscribirMatriz(Inversa(B), "InversaB.txt");
22.
23.     printf("Matriz A:\n");
24.     printMatriz(LeerMatriz("MatrizA.txt"));
25.     printf("\nMatriz B:\n");
26.     printMatriz(LeerMatriz("MatrizB.txt"));
27.     printf("\nSuma de matrices:\n");
28.     printMatriz(LeerMatriz("Suma.txt"));
29.     printf("\nResta de matrices:\n");
30.     printMatriz(LeerMatriz("Resta.txt"));
31.     printf("\nMultiplicacion de matrices:\n");
32.     printMatriz(LeerMatriz("Multiplicacion.txt"));
33.     printf("\nTranspuesta de A:\n");
34.     printMatriz(LeerMatriz("TranspuestaA.txt"));
35.     printf("\nTranspuesta de B:\n");
36.     printMatriz(LeerMatriz("TranspuestaB.txt"));
37.     printf("\nInversa de A:\n");
38.     printMatriz(LeerMatriz("InversaA.txt"));
39.     printf("\nInversa de B:\n");
40.     printMatriz(LeerMatriz("InversaB.txt"));
41.     clock_t tFin = clock();
42.     printf("\nTiempo: %0.7fs\n", (double)(tFin - tInicio) / CLOCKS_PER_SEC);
43.     return 0;
44. }
```

## Compilación y ejecución del programa con 6 procesos:

```
C:\Users\Edmundo J Sanchez M\Desktop\S0\P3\Matriz6ProcesosSustitucion>gcc ElegirProcesoM.c -o ElegirProcesoM.exe
C:\Users\Edmundo J Sanchez M\Desktop\S0\P3\Matriz6ProcesosSustitucion>gcc Matriz6ProcSust.c -o Matriz6ProcSust.exe
C:\Users\Edmundo J Sanchez M\Desktop\S0\P3\Matriz6ProcesosSustitucion>Matriz6ProcSust.exe
Matriz A:
4.000 2.000 9.000 9.000 2.000 5.000 3.000
1.000 7.000 7.000 1.000 8.000 5.000 1.000
7.000 8.000 7.000 7.000 5.000 3.000 2.000
9.000 8.000 7.000 8.000 1.000 6.000 2.000
3.000 3.000 1.000 1.000 7.000 6.000 4.000
9.000 0.000 8.000 4.000 0.000 7.000 0.000
3.000 1.000 0.000 5.000 7.000 7.000 2.000

Matriz B:
6.000 8.000 0.000 4.000 7.000 7.000 9.000
6.000 8.000 4.000 6.000 5.000 0.000 6.000
5.000 1.000 3.000 8.000 3.000 1.000 5.000
0.000 3.000 7.000 7.000 1.000 7.000 0.000
4.000 1.000 9.000 6.000 0.000 3.000 8.000
0.000 8.000 4.000 3.000 0.000 7.000 7.000
1.000 8.000 5.000 2.000 9.000 9.000 3.000

Suma de matrices:
10.000 10.000 9.000 13.000 9.000 12.000 12.000
7.000 15.000 11.000 7.000 13.000 5.000 7.000
12.000 9.000 10.000 15.000 8.000 4.000 7.000
9.000 11.000 14.000 15.000 2.000 13.000 2.000
7.000 4.000 10.000 7.000 7.000 9.000 12.000
9.000 8.000 12.000 7.000 0.000 14.000 7.000
4.000 9.000 5.000 7.000 16.000 16.000 5.000

Resta de matrices:
-2.000 -6.000 9.000 5.000 -5.000 -2.000 -6.000
-5.000 -1.000 3.000 -5.000 3.000 5.000 -5.000
2.000 7.000 4.000 -1.000 2.000 2.000 -3.000
9.000 5.000 0.000 1.000 0.000 -1.000 2.000
-1.000 2.000 -8.000 -5.000 7.000 3.000 -4.000
9.000 -8.000 4.000 1.000 0.000 0.000 -7.000
2.000 -7.000 -5.000 3.000 -2.000 -2.000 -1.000

Multiplicacion de matrices:
92.000 150.000 151.000 196.000 101.000 168.000 153.000
116.000 130.000 153.000 174.000 73.000 89.000 188.000
147.000 193.000 169.000 224.000 135.000 159.000 213.000
143.000 232.000 152.000 224.000 150.000 189.000 220.000
73.000 139.000 129.000 113.000 76.000 128.000 160.000
94.000 148.000 80.000 149.000 91.000 148.000 170.000
54.000 126.000 140.000 120.000 49.000 144.000 144.000

Transpuesta de A:
4.000 1.000 7.000 9.000 3.000 9.000 3.000
2.000 7.000 8.000 8.000 3.000 0.000 1.000
9.000 7.000 7.000 7.000 1.000 8.000 0.000
9.000 1.000 7.000 8.000 1.000 4.000 5.000
2.000 8.000 5.000 1.000 7.000 0.000 7.000
5.000 5.000 3.000 6.000 6.000 7.000 7.000
3.000 1.000 2.000 2.000 4.000 0.000 2.000

Transpuesta de B:
6.000 6.000 5.000 0.000 4.000 0.000 1.000
8.000 8.000 1.000 3.000 1.000 8.000 8.000
0.000 4.000 3.000 7.000 9.000 4.000 5.000
4.000 6.000 8.000 7.000 6.000 3.000 2.000
7.000 5.000 3.000 1.000 0.000 0.000 9.000
7.000 0.000 1.000 7.000 3.000 7.000 9.000
9.000 6.000 5.000 0.000 8.000 7.000 3.000

Inversa de A:
-0.085 -0.095 0.155 -0.068 0.056 0.096 -0.024
-0.047 0.060 -0.056 0.149 -0.016 -0.079 -0.021
0.070 0.052 0.016 -0.070 0.004 0.048 -0.085
0.056 -0.033 0.026 0.016 -0.111 -0.058 0.111
-0.030 0.006 0.160 -0.161 -0.010 0.032 0.064
-0.002 0.082 -0.232 0.158 -0.014 -0.003 0.064
0.124 -0.111 -0.015 -0.003 0.285 -0.062 -0.182

Inversa de B:
0.328 0.131 -0.420 0.251 0.141 -0.302 -0.216
0.003 0.125 -0.102 0.048 -0.054 0.042 -0.043
-0.046 0.054 -0.106 0.018 0.111 -0.054 0.039
-0.102 -0.033 0.253 -0.009 -0.118 0.107 0.014
-0.123 -0.050 0.206 -0.148 -0.044 0.026 0.182
0.164 -0.068 -0.133 0.134 0.037 -0.074 -0.061
-0.098 -0.092 0.202 -0.196 0.012 0.155 0.081

Tiempo: 0.212000s
```



## Compilación y ejecución del programa de forma secuencial:

```
C:\Users\Edmundo J Sanchez M\Desktop\S0\P3\Matriz6ProcesosSustitucion>gcc OpSecuencial.c -o OpSecuencial.exe
```

```
C:\Users\Edmundo J Sanchez M\Desktop\S0\P3\Matriz6ProcesosSustitucion>OpSecuencial.exe
```

Matriz A:

3.000	8.000	8.000	0.000	1.000	0.000	4.000
9.000	5.000	0.000	2.000	9.000	8.000	0.000
0.000	6.000	3.000	2.000	8.000	6.000	5.000
7.000	5.000	2.000	7.000	3.000	4.000	4.000
7.000	7.000	9.000	0.000	5.000	6.000	2.000
3.000	9.000	8.000	3.000	4.000	7.000	7.000
4.000	7.000	1.000	7.000	3.000	4.000	5.000

Matriz B:

7.000	8.000	9.000	3.000	8.000	8.000	2.000
2.000	7.000	5.000	5.000	1.000	1.000	1.000
1.000	4.000	4.000	3.000	7.000	6.000	2.000
4.000	5.000	2.000	3.000	1.000	6.000	9.000
3.000	9.000	8.000	9.000	7.000	9.000	0.000
1.000	0.000	3.000	8.000	4.000	3.000	2.000
9.000	2.000	5.000	7.000	9.000	8.000	7.000

Suma de matrices:

10.000	16.000	17.000	3.000	9.000	8.000	6.000
11.000	12.000	5.000	7.000	10.000	9.000	1.000
1.000	10.000	7.000	5.000	15.000	12.000	7.000
11.000	10.000	4.000	10.000	4.000	10.000	13.000
10.000	16.000	17.000	9.000	12.000	15.000	2.000
4.000	9.000	11.000	11.000	8.000	10.000	9.000
13.000	9.000	6.000	14.000	12.000	12.000	12.000

Resta de matrices:

-4.000	0.000	-1.000	-3.000	-7.000	-8.000	2.000
7.000	-2.000	-5.000	-3.000	8.000	7.000	-1.000
-1.000	2.000	-1.000	-1.000	1.000	0.000	3.000
3.000	0.000	0.000	4.000	2.000	-2.000	-5.000
4.000	-2.000	1.000	-9.000	-2.000	-3.000	2.000
2.000	9.000	5.000	-5.000	0.000	4.000	5.000
-5.000	5.000	-4.000	0.000	-6.000	-4.000	-2.000

Multiplicacion de matrices:

84.000	129.000	127.000	110.000	131.000	121.000	58.000
116.000	198.000	206.000	203.000	174.000	194.000	57.000
98.000	146.000	153.000	200.000	154.000	166.000	77.000
138.000	169.000	166.000	160.000	155.000	186.000	122.000
111.000	190.000	202.000	190.000	203.000	196.000	65.000
141.000	184.000	198.000	228.000	211.000	212.000	121.000
129.000	157.000	150.000	165.000	135.000	166.000	123.000

Transpuesta de A:

3.000	9.000	0.000	7.000	7.000	3.000	4.000
8.000	5.000	6.000	5.000	7.000	9.000	7.000
8.000	0.000	3.000	2.000	9.000	8.000	1.000
0.000	2.000	2.000	7.000	0.000	3.000	7.000
1.000	9.000	8.000	3.000	5.000	4.000	3.000
0.000	8.000	6.000	4.000	6.000	7.000	4.000
4.000	0.000	5.000	4.000	2.000	7.000	5.000

Transpuesta de B:

7.000	2.000	1.000	4.000	3.000	1.000	9.000
8.000	7.000	4.000	5.000	9.000	0.000	2.000
9.000	5.000	4.000	2.000	8.000	3.000	5.000
3.000	5.000	3.000	3.000	9.000	8.000	7.000
8.000	1.000	7.000	1.000	7.000	4.000	9.000
8.000	1.000	6.000	6.000	9.000	3.000	8.000
2.000	1.000	2.000	9.000	0.000	2.000	7.000

Inversa de A:

0.141	0.164	-0.095	0.222	-0.239	0.146	-0.304
0.045	0.040	-0.083	-0.345	0.083	-0.069	0.387
-0.097	-0.201	0.099	0.046	0.293	-0.164	0.053
-0.193	-0.254	0.110	-0.035	0.375	-0.306	0.351
0.042	-0.048	0.228	0.118	0.062	-0.218	-0.076
-0.186	0.033	-0.125	-0.158	0.053	0.201	0.098
0.237	0.211	-0.018	0.401	-0.589	0.411	-0.632

Inversa de B:

-0.056	0.035	-0.150	-0.061	0.083	-0.174	0.183
-0.201	0.187	0.128	-0.037	0.081	-0.272	0.120
0.428	-0.116	-0.122	0.088	-0.216	0.439	-0.309
-0.120	0.055	-0.043	-0.026	0.081	0.009	0.069
-0.147	0.141	0.266	-0.122	-0.061	-0.144	0.144
0.073	-0.260	-0.134	0.100	0.160	0.043	-0.087
0.049	0.046	0.097	0.095	-0.161	0.115	-0.060

Tiempo: 0.531000s



Vemos que el tiempo fue considerablemente mayor que en Linux, esto es debido a que Windows no les da la misma prioridad a los procesos que Linux, y a que su inicialización tarda más, ahora analizando ambos tipos de ejecución, tenemos el mismo resultado que el que obtuvimos de Linux, el cual es que la sustitución de código con procesos es más rápida que el secuencial.

### 3. Análisis de la practica

#### 3.1. Linux

Con esta práctica pudimos observar el uso de procesos que se ejecutan de manera concurrente o también se pueden ejecutar un proceso tras otro. Se hizo uso de llamadas al sistema que controlaban los procesos, donde podemos observar las siguientes funciones:

- Crear un nuevo proceso.
- Carga de un proceso dentro de otro.
- Finalizar un proceso.
- Esperar a que termine un proceso.

Se debe tener cuidado como se invoca **fork()** para los procesos ya que se pueden varios procesos del mismo padre o el proceso padre genera un hijo y así sucesivamente los cuales finalizan con la llamada al sistema **exit(0)**. También al hacer uso de la llamada **execv(const char\* path, char\* const argv[])** se debe tener cuidado con la ruta que se manda, ya que se tiene que mandar la del archivo ejecutable y no la del código fuente, en caso de que se mande la del código fuente el valor de retorno de la llamada al sistema **execv** es -1 que significa que fallo la llamada al sistema.

#### 3.2. Windows

La creación de procesos en Windows es notablemente más compleja: se necesita un conocimiento mínimo del uso de la función **CreateProcess**, y además los argumentos que se le mandan a la función. Esto es muy importante porque cualquier error en esta parte puede provocar un mal funcionamiento de la aplicación.

Un aspecto que hay que tener en cuenta es la similitud entre el primer y segundo argumento de la función **CreateProcess**, ya que pueden llegar a causar una confusión y estos dos argumentos, se podría decir, que son los más importantes de todos.

Otro aspecto importante es que por cada proceso que se invoque en el programa a su vez se deben de cerrar con la función **CloseHandle(\*especificación de proceso\*)**.

## 4. Observaciones

### 4.1. Linux

- Al momento de llamar a **fork()**, se divide (de ahí el nombre de la llamada) en dos a partir de ese punto la ejecución del programa, la primera división es el padre cuyo valor de retorno es cero, y la otra división es el proceso hijo con valor de retorno positivo (el PID del hijo).
- Debido a lo anterior, hay que cuidar mucho en donde colocamos los **exit(0)**, dependiendo el comportamiento que deseemos.
- La llamada **wait(status)** pone en espera el proceso actual hasta que alguno de sus hijos termine. Devuelve el PID del proceso que termino y copia en status el valor de retorno de dicho proceso. Si no había procesos hijo, se ignora. Por lo que, por ejemplo, si queremos crear cinco procesos hijo en la memoria, primero llamamos 5 veces a **fork()** y luego 5 veces a **wait()**, y no de forma intercalada.
- En el punto de la creación del árbol de procesos, nunca existió el 100 % del árbol completo en ningún momento, pues algunos procesos padre terminaban su ejecución mientras sus procesos hijo aun seguían. Nos dimos cuenta de esto por el orden de los eventos que se imprimían en las salidas del programa.
- Es un poco más rápido crear aplicaciones concurrentes usando copia exacta que hacerlo secuencialmente, pero todavía es más rápido usar procesos por sustitución, pues no tenemos que copiar todo el código actual, solo el del ejecutable que se necesita.
- Al hacer uso de la llamada **execv** en el parámetro de la ruta se tiene que mandar la ruta el archivo ejecutable, también se debe incluir en el primer elemento del arreglo **argv**.
- Si queremos llamar a otro proceso usando **execv** y que el proceso principal también siga corriendo, hay que hacerlo dentro de un proceso creado por copia exacta, porque de no hacerlo así el proceso original se destruye.
- Con la llamada **fork()** podemos crear a partir de un proceso padre más de un proceso hijo. Y con esto ejecutar concurrentemente varios procesos.
- Al crear procesos de manera concurrente, en algún momento los procesos se llegan a juntar, por lo que el sistema podría llegar a confundirse.
- Los PID's asignados por Linux tienden a ser consecutivos.

### 4.2. Windows

- En Windows no es posible crear procesos por copia exacta de código.
- Cuando usemos la llamada **CreateProcess**, en el primer argumento usualmente va el nombre del ejecutable, aunque se puede omitir. El segundo argumento es una cadena en donde van todos los argumentos que le vamos a pasar al ejecutable, comenzando justamente con el nombre del ejecutable, separando con espacios los demás argumentos y usando comillas dobles en caso de que un argumento contenga espacios. Es decir, es como unir todos los elementos del arreglo **argv** que usamos en Linux separándolos por espacios.

- Al crear nuevos procesos, estos se crean como hilos, es decir, el programa original seguirá su ejecución normal a diferencia de Linux.
- En Windows también podemos esperar a que un proceso hijo termine usando la llamada **WaitForSingleObject**, pero aquí necesitaremos el ID del proceso, mientras que en Linux no.
- Windows nos obliga también a liberar los recursos del proceso hijo una vez que ya terminó su ejecución.
- La ejecución de procesos concurrentes es más desordenada que en Linux.
- Tanto en Linux como en Windows podemos usar rutas absolutas y relativas para hacer referencia a los ejecutables.
- Los PID's asignados por Windows parecen ser asignados de forma aleatoria

## 5. Conclusiones

Tanto en Linux como en Windows se cuentan con distintas llamadas al sistema para la creación y manipulación de procesos, ambos sistemas crean, cargan, finalizan y esperan procesos. Estas llamadas al sistema normalmente devuelven cero si se ejecutaron con éxito o -1 en caso de error y su forma de invocación cambia en ambos sistemas operativos, sin embargo, es el mismo funcionamiento igual que en la práctica anterior.

Pudimos observar que las llamadas al sistema **getpid()** y **getppid()** simulan la misma función que el comando "**ps**" o "**ps -fea**", en este caso son similares a una columna. Y a diferencia de los comandos se pueden ejecutar varias llamadas al sistema en el mismo código, no hay ninguna restricción respecto a sus veces de invocación.

La creación de procesos es notablemente más difícil en el sistema operativo Windows por todas las consideraciones antes descritas.

Hay funciones similares entre los dos sistemas operativos, una de ellas es **wait()** y **getpid()** en el sistema operativo Linux, y **WaitForSingleObject()** y **GetCurrentProcessId()** en el sistema operativo Windows.