

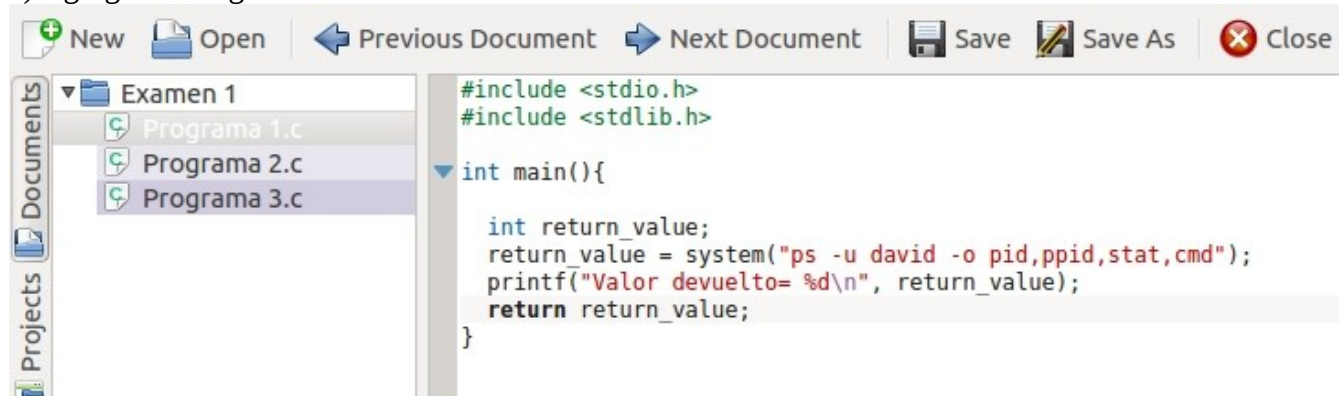
Altamirano Peralta David

Arquitectura Cliente Servidor

Examen 1

1.- Programa 1

a) Agrega el código fuente.



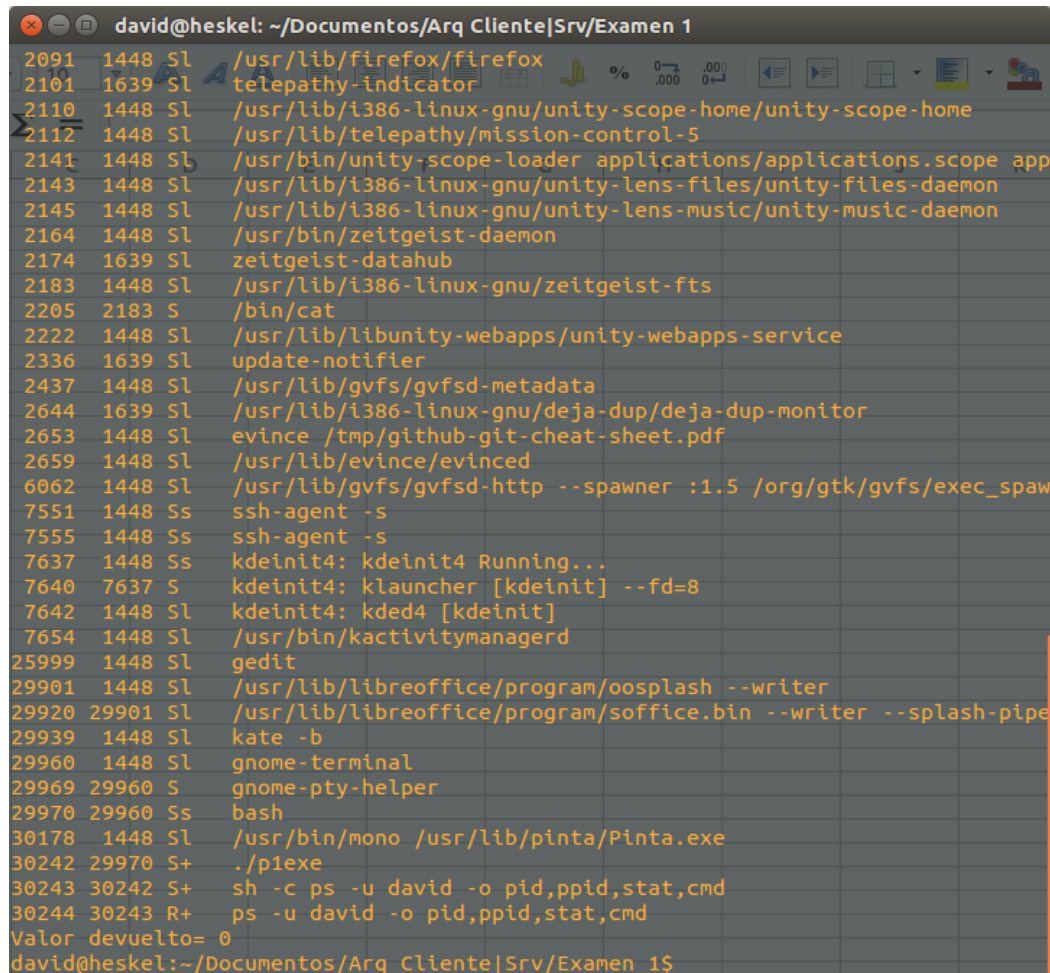
The screenshot shows a code editor window titled 'Examen 1'. The left sidebar shows a file tree with 'Programa 1.c', 'Programa 2.c', and 'Programa 3.c'. The main editor area displays the following C code:

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

int main(){

    int return_value;
    return_value = system("ps -u david -o pid,ppid,stat,cmd");
    printf("Valor devuelto= %d\n", return_value);
    return return_value;
}
```

b) Agrega la captura de pantalla con la salida de la ejecución.



The screenshot shows a terminal window titled 'david@heskel: ~/Documentos/Arq Cliente|Srv/Examen 1'. The terminal displays the output of the 'ps' command, showing a list of running processes with their PIDs, PPIDs, and states. The output is as follows:

```
2091 1448 SL /usr/lib/firefox/firefox
2101 1639 SL telepathy-indicator
2110 1448 SL /usr/lib/i386-linux-gnu/unity-scope-home/unity-scope-home
2112 1448 SL /usr/lib/telepathy/mission-control-5
2141 1448 SL /usr/bin/unity-scope-loader applications/applications.scope app
2143 1448 SL /usr/lib/i386-linux-gnu/unity-lens-files/unity-files-daemon
2145 1448 SL /usr/lib/i386-linux-gnu/unity-lens-music/unity-music-daemon
2164 1448 SL /usr/bin/zeitgeist-daemon
2174 1639 SL zeitgeist-datahub
2183 1448 SL /usr/lib/i386-linux-gnu/zeitgeist-fts
2205 2183 S /bin/cat
2222 1448 SL /usr/lib/libunity-webapps/unity-webapps-service
2336 1639 SL update-notifier
2437 1448 SL /usr/lib/gvfs/gvfsd-metadata
2644 1639 SL /usr/lib/i386-linux-gnu/deja-dup/deja-dup-monitor
2653 1448 SL evince /tmp/github-git-cheat-sheet.pdf
2659 1448 SL /usr/lib/evince/evince
6062 1448 SL /usr/lib/gvfs/gvfsd-http --spawner :1.5 /org/gtk/gvfs/exec_spaw
7551 1448 Ss ssh-agent -s
7555 1448 Ss ssh-agent -s
7637 1448 Ss kdeinit4: kdeinit4 Running...
7640 7637 S kdeinit4: klauncher [kdeinit] --fd=8
7642 1448 SL kdeinit4: kded4 [kdeinit]
7654 1448 SL /usr/bin/kactivitymanagerd
25999 1448 SL gedit
29901 1448 SL /usr/lib/libreoffice/program/oosplash --writer
29920 29901 SL /usr/lib/libreoffice/program/soffice.bin --writer --splash-pipe
29939 1448 SL kate -b
29960 1448 SL gnome-terminal
29969 29960 S gnome-ptty-helper
29970 29960 Ss bash
30178 1448 SL /usr/bin/mono /usr/lib/pinta/Pinta.exe
30242 29970 S+ ./p1exe
30243 30242 S+ sh -c ps -u david -o pid,ppid,stat,cmd
30244 30243 R+ ps -u david -o pid,ppid,stat,cmd
Valor devuelto= 0
david@heskel:~/Documentos/Arq Cliente|Srv/Examen 1$
```

c) Explica detalladamente qué hace el programa y explica también por que system() devuelve ese valor.

```
#include <stdio.h>  //headers de ANSI C
#include <stdlib.h>

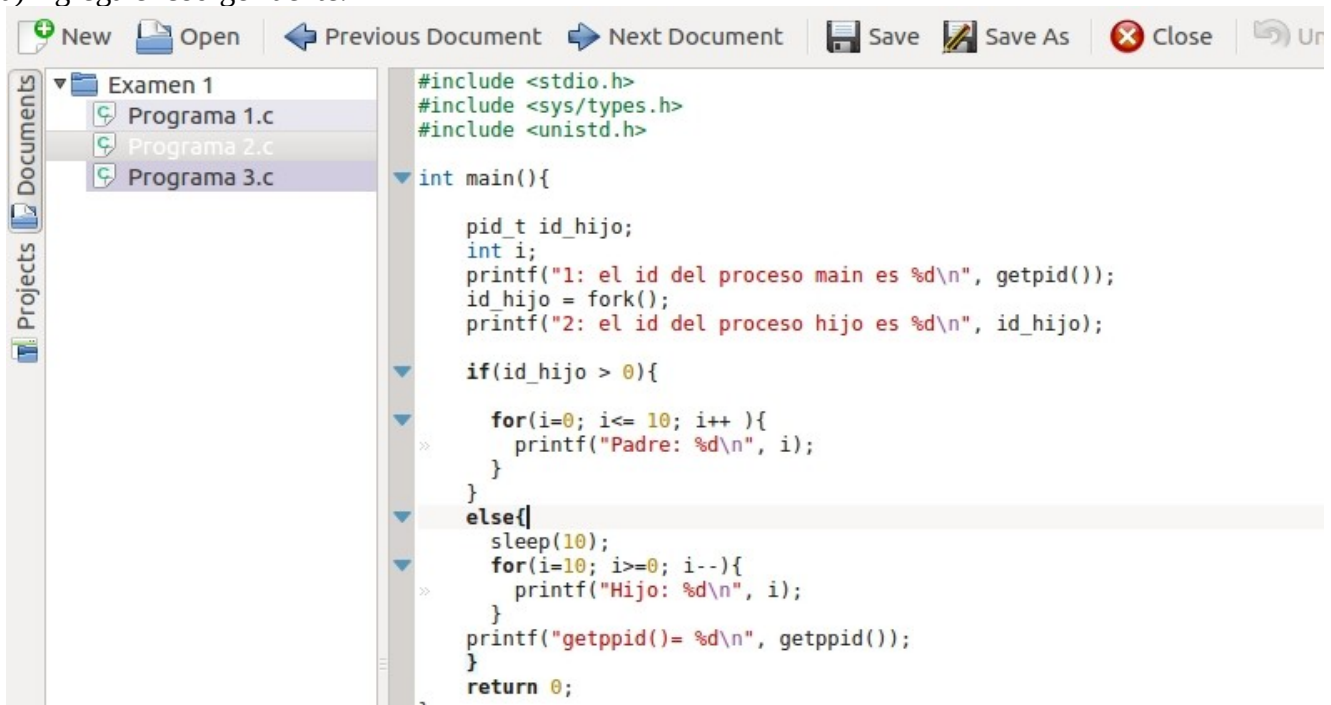
int main(){

    int return_value;      //variable de tipo entero para almacenar el valor de retorno de
                           //System
    return_value = system("ps -u david -o pid,ppid,stat,cmd"); //se le asigna la operación
                           //system a return_value y se ejecuta la instrucción "ps" para el
                           //usuario local con los parámetros de pid, ppid, stat y cmd
    printf("Valor devuelto= %d\n", return_value);      //imprime el valor de return_value el
                           //cual es 0
    return return_value;  //regresa el valor 0 de return_value y la ejecución de main
                           //termina correctamente.
}

/*El valor que devuelve system es cero por que la ejecución de "ps -u david -o
pid,ppid,stat,cmd" terminó correctamente y cuando un programa termina correctamente devuelve
el valor 0 */
```

2.- Programa 2

a) Agrega el código fuente.

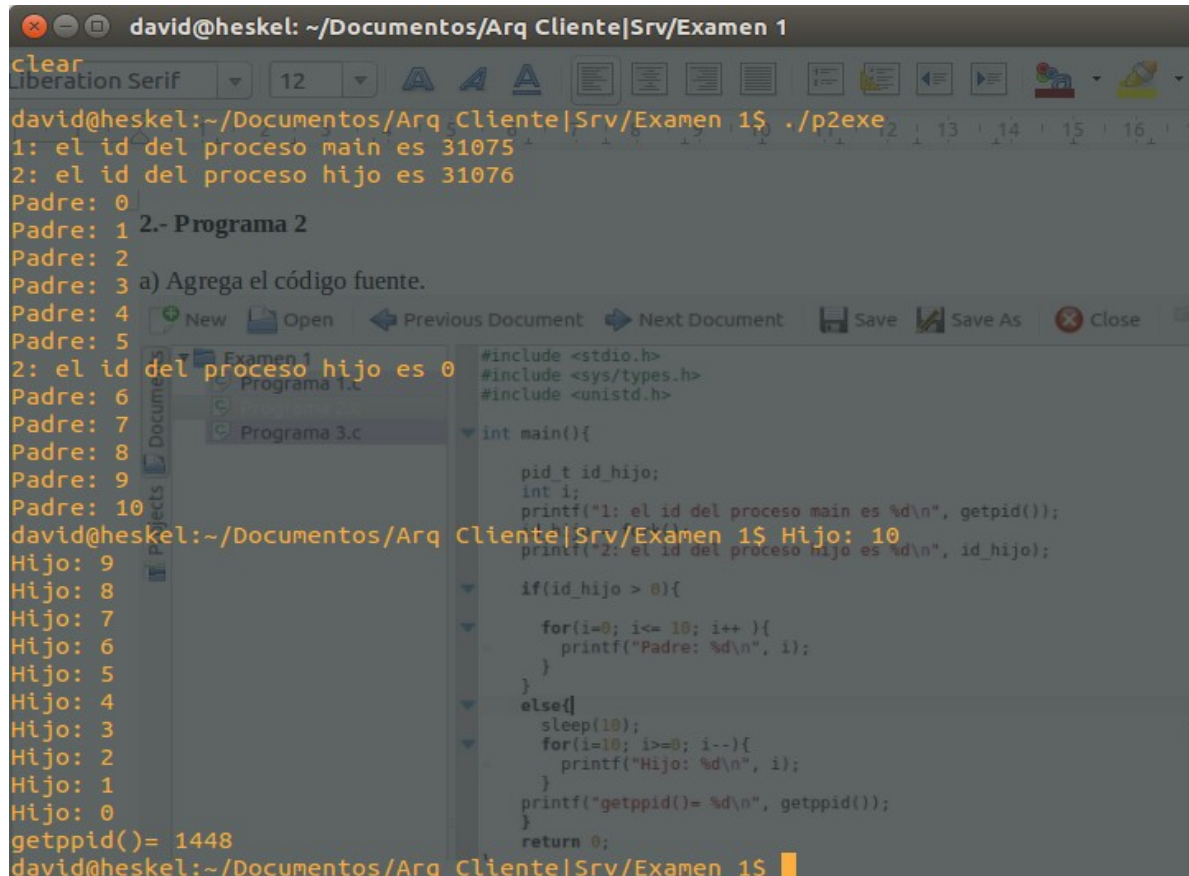


```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
    pid_t id_hijo;
    int i;
    printf("1: el id del proceso main es %d\n", getpid());
    id_hijo = fork();
    printf("2: el id del proceso hijo es %d\n", id_hijo);

    if(id_hijo > 0){
        for(i=0; i<= 10; i++){
            printf("Padre: %d\n", i);
        }
    }
    else{
        sleep(10);
        for(i=10; i>=0; i--){
            printf("Hijo: %d\n", i);
        }
        printf("getppid()= %d\n", getppid());
    }
    return 0;
}
```

b) Agrega la captura de pantalla con la salida de la ejecución



```
clear
david@heskel: ~/Documentos/Arq Cliente/Srv/Examen 1$ ./p2exe
1: el id del proceso main es 31075
2: el id del proceso hijo es 31076
Padre: 0
Padre: 1
Padre: 2
Padre: 3
Padre: 4
Padre: 5
Padre: 6
Padre: 7
Padre: 8
Padre: 9
Padre: 10
Hijo: 10
Hijo: 9
Hijo: 8
Hijo: 7
Hijo: 6
Hijo: 5
Hijo: 4
Hijo: 3
Hijo: 2
Hijo: 1
Hijo: 0
getppid()= 1448
david@heskel: ~/Documentos/Arq Cliente/Srv/Examen 1$
```

c) Explica detalladamente qué hace el programa y explica también por qué cuando el proceso hijo termina, no regresa el prompt del sistema

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){

    pid_t id_hijo; //Crea una estructura del tipo pid_t que sirve para identificar procesos
    int i;          //Variable de tipo int
    printf("1: el id del proceso main es %d\n", getpid()); // Se imprime el ID del proceso padre u
proceso original
    id_hijo = fork(); // Se crea el proceso hijo identico al proceso
padre
    printf("2: el id del proceso hijo es %d\n", id_hijo); // Se imprime el ID del proceso hijo

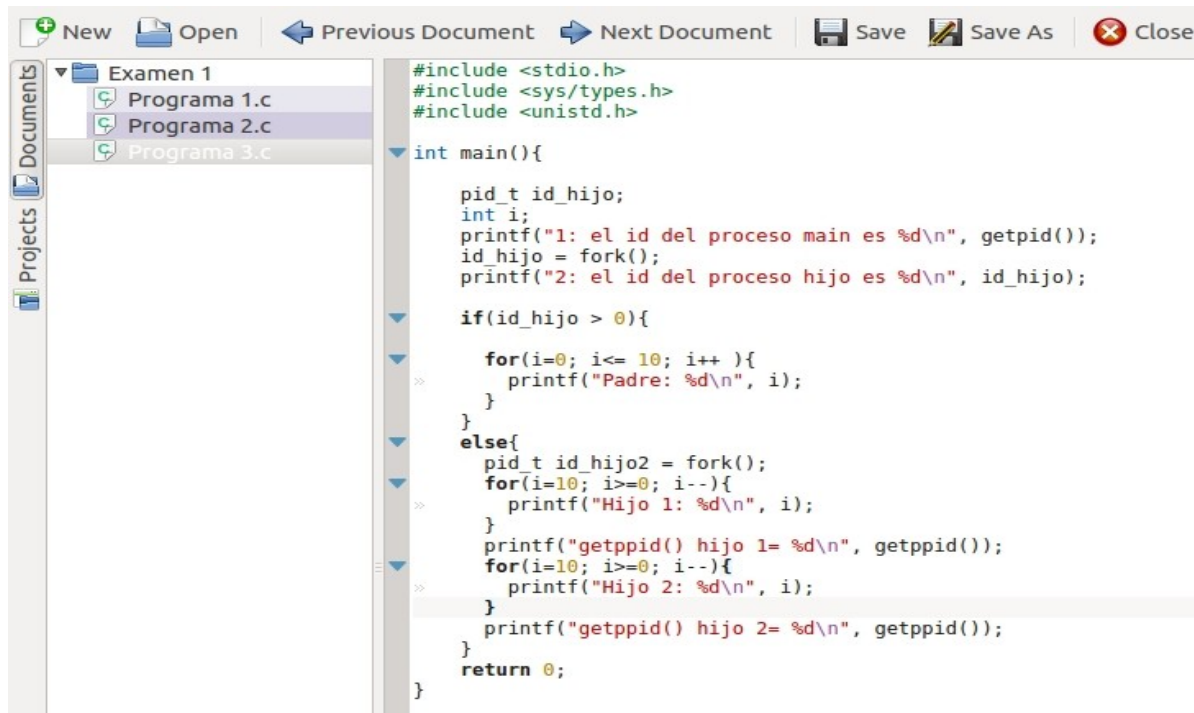
    if(id_hijo > 0){ //Esta condición verifica que el proceso hijo se crea correctamente y entra en la
condición

        for(i=0; i<= 10; i++){
            printf("Padre: %d\n", i); //Se imprime el Padre n desde 0 hasta que i toma el valor de 10 y
            //termina el proceso padre
        }
    }
    else{
        sleep(10); // El proceso espera 10 segundos
        for(i=10; i>=0; i--){
            printf("Hijo: %d\n", i); //Se imprime el Hijo: n desde 10 hasta que i toma el valor de 0
        }
        printf("getppid()= %d\n", getppid()); // se imprime el ID del proceso padre del padre es decir del
proceso init, por que el proceso padre ya terminó
    }
    return 0;
}

/*El proceso no termina hasta que se da un enter dado que el proceso init nunca termina de ejecutarse y
solo puede terminarse
la ejecución del proceso al presionar enter. */
```

3.- Proceso 3

a) Agrega el código fuente.



The screenshot shows a C code editor with a menu bar (New, Open, Previous Document, Next Document, Save, Save As, Close) and a sidebar with 'Documents' and 'Projects' sections. The 'Documents' section shows a folder 'Examen 1' containing three files: 'Programa 1.c', 'Programa 2.c', and 'Programa 3.c'. The main editor area displays the following C code:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
    pid_t id_hijo;
    int i;
    printf("1: el id del proceso main es %d\n", getpid());
    id_hijo = fork();
    printf("2: el id del proceso hijo es %d\n", id_hijo);

    if(id_hijo > 0){
        for(i=0; i<= 10; i++){
            printf("Padre: %d\n", i);
        }
    }
    else{
        pid_t id_hijo2 = fork();
        for(i=10; i>=0; i--){
            printf("Hijo 1: %d\n", i);
        }
        printf("getppid() hijo 1= %d\n", getppid());
        for(i=10; i>=0; i--){
            printf("Hijo 2: %d\n", i);
        }
        printf("getppid() hijo 2= %d\n", getppid());
    }
    return 0;
}
```

b) Agrega la captura de pantalla con la salida de la ejecución.

c) Explica detalladamente que hace el programa y modificalo para que cada ciclo se imprima solo una vez.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){

    pid_t id_hijo; //Crea una estructura del tipo pid_t que sirve para identificar procesos
    int i;         //variable de tipo int
    printf("1: el id del proceso main es %d\n", getpid()); // Se imprime el ID del proceso padre u
proceso original
    id_hijo = fork(); // Se crea el proceso hijo con fork () identico al proceso padre
    printf("2: el id del proceso hijo es %d\n", id_hijo); // Se imprime el ID del proceso hijo

    if(id_hijo > 0){ //se entra a la condifición dado que id_hijo es diferente de cero

        for(i=0; i<= 10; i++){
            printf("Padre: %d\n", i); //Se imprime el Padre n desde 0 hasta que i toma el valor de 10 y
termina el proceso padre
        }
    }
    else{
        pid_t id_hijo2 = fork(); // Se crea un proceso hijo 2 en el hijo 1
        for(i=10; i>=0; i--){
            printf("Hijo 1: %d\n", i); //Se imprime el Hijo: n desde 10 hasta que i toma el valor de 0
        }
        printf("getppid() hijo 1= %d\n", getppid()); //dado que el proceso padre ya termino se trae el
valor de proceso init por que el proceso padre ya no existe
        for(i=10; i>=0; i--){
            printf("Hijo 2: %d\n", i); //Se imprime el Hijo: n desde 10 hasta que i toma el valor de 0
        }
        printf("getppid() hijo 2= %d\n", getppid()); //lo mismo para el proceso hijo 2, el proceso hijo
                                                    //1 ya terminó al igual al igual que el proceso
                                                    //padre de manera que devuelve el valor ID del
                                                    //proceso init
    }
    return 0;
}
```

Programa modificado:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){

    pid_t id_hijo; //Crea una estructura del tipo pid_t que sirve para identificar procesos
    pid_t id_hijo2; // = fork();
    int i;         //variable de tipo int
    printf("1: el id del proceso main es %d\n", getpid()); // Se imprime el ID del proceso padre u
proceso original
    id_hijo = fork(); // Se crea el proceso hijo identico al proceso
padre
    //id_hijo2 = fork();
    printf("2: el id del proceso hijo es %d\n", id_hijo); // Se imprime el ID del proceso hijo

    if(id_hijo > 0){

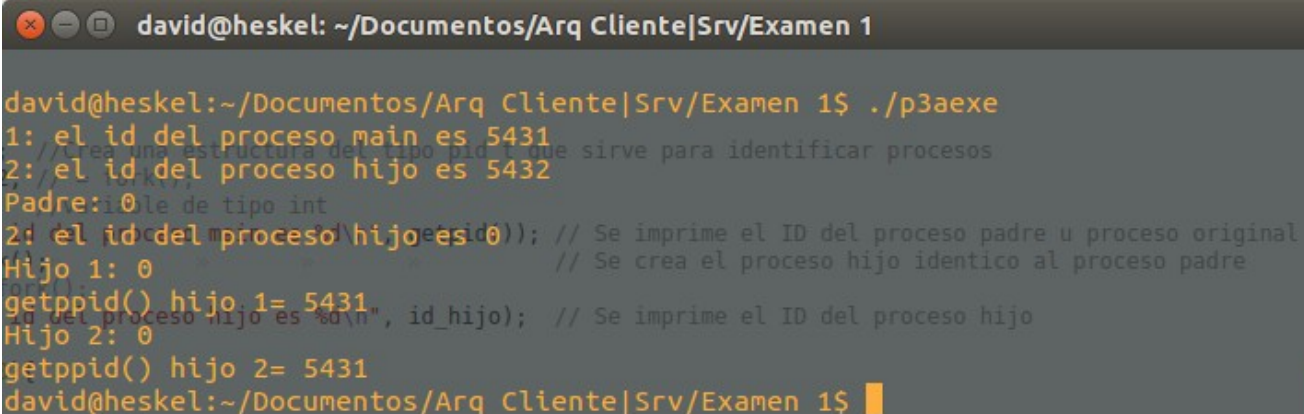
        for(i=0; i<= 0; i++){
            printf("Padre: %d\n", i); //Se imprime el Padre n desde 0 hasta que i toma el valor de 10 y
termina el proceso padre
        }
    }
    else{
```

```

for(i=0; i>=0; i--){
    printf("Hijo 1: %d\n", i); //Se imprime el Hijo: n desde 10 hasta que i toma el valor de 0
}
printf("getppid() hijo 1= %d\n", getppid()); //dado que el proceso padre ya termino se trae el
valor de proceso init
for(i=0; i>=0; i--){
    printf("Hijo 2: %d\n", i); //Se imprime el Hijo: n desde 10 hasta que i toma el valor de 0
}
printf("getppid() hijo 2= %d\n", getppid()); // lo mismo para el proceso hijo 2, el proceso hijo
1 ya terminó al igual
    id_hijo2 = fork(); // Se crea un proceso hijo en el hijo al final de los bucles para que se cree
pero no realiza nada ya que está después de los bucles.

}
// al igual que el proceso padre de manera que devuelve
el valor ID del
return 0;
}
//proceso init

```



```

david@heskel:~/Documentos/Arq Cliente|Srv/Examen 1$ ./p3aexe
1: el id del proceso main es 5431
2: el id del proceso hijo es 5432
Padre: 0
2: el id del proceso hijo es 0
Hijo 1: 0
getppid() hijo 1= 5431
Hijo 2: 0
getppid() hijo 2= 5431
david@heskel:~/Documentos/Arq Cliente|Srv/Examen 1$

```