Universidad Autónoma de Baja California  
Facultad de Ciencias Químicas e Ingeniería

Nombre: Gómez Cárdenas Emmanuel Alberto  
Materia: Sistemas Operativos  
Maestro: Carlos Fco. Alvarez Salgado

Procesos

1. Observa el siguiente código y escribe la jerarquía de procesos resultante.

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main (int argc, char \*argv[]) {

    int num;

    pid\_t pid;

    for (num = 0; num < 3; num++) {

        pid = fork();

        printf ("Soy el proceso de PID %d y mi padre tiene %d de PID.\n",getpid(), getppid());

        if (pid!= 0)    break;

        srandom(getpid());

        sleep (random() %3);

    }

    if (pid!= 0)    printf ("Fin del proceso de PID %d.\n", wait (NULL));

    return 0;

}

Padre -> Hijo1 -> Hijo2 -> Hijo3

Ahora compila y ejecuta el código para comprobarlo. Contesta a las siguientes preguntas:

* **¿Por qué aparecen mensajes repetidos?**  
  Porque al momento de crear el proceso hijo con fork, este ejecuta el mismo código que el padre. Por esto se imprimen los PID’s de ambos procesos y al repetir el bloque for se siguen ejecutando más forks.

Presta atención al orden de terminación de los procesos

* **¿Qué observas?**El primer proceso en terminar la ejecución es el último proceso hijo creado.
* **¿Por qué?**Funciona como si fuera una función recursiva donde la última función llamada es la primera en acabar (Los padres esperan a que terminen los hijos para poder seguir ejecutando las instrucciones).

2. Observa el siguiente código y escribe la jerarquía de procesos resultante.

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main (int argc, char argv[]) {

    int num;

    pid\_t pid;

    srandom(getpid());

    for (num = 0; num < 3; num++) {

        pid = fork();

        printf ("Soy el proceso de PID %d y mi padre tiene %d de PID.\n", getpid(), getppid());

        if (pid == 0)   break;

    }

    if (pid == 0)   sleep(random() %5);

    else for (num = 0; num < 3; num++)

    printf ("Fin del proceso de PID %d.\n", wait (NULL));

    return 0;

}

Padre -> Hijo1; Padre -> Hijo2; Padre -> Hijo3

* **Ahora compila y ejecuta el código para comprobarlo. Presta atención al orden de terminación de los procesos ¿qué observas?**En este caso los primeros procesos hijos creados son los primeros en terminar.
* **¿Por qué?**Por la condición if (pid==0), cuando creamos un proceso hijo, este se pone a dormir (sleep) y cuando termina el primer bucle for pasamos al último bucle el cual solo espera que los hijos salgan del sleep para terminar de ejecutar las instrucciones.

3. Dibuja la estructura del árbol de procesos que obtendríamos al ejecutar el siguiente fragmento de código:

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main (int argc, char \*argv[]) {

    int num;

    pid\_t nuevo;

    for (num = 0; num < 2; num++) {

        nuevo = fork();

        if (nuevo == 0)  break;

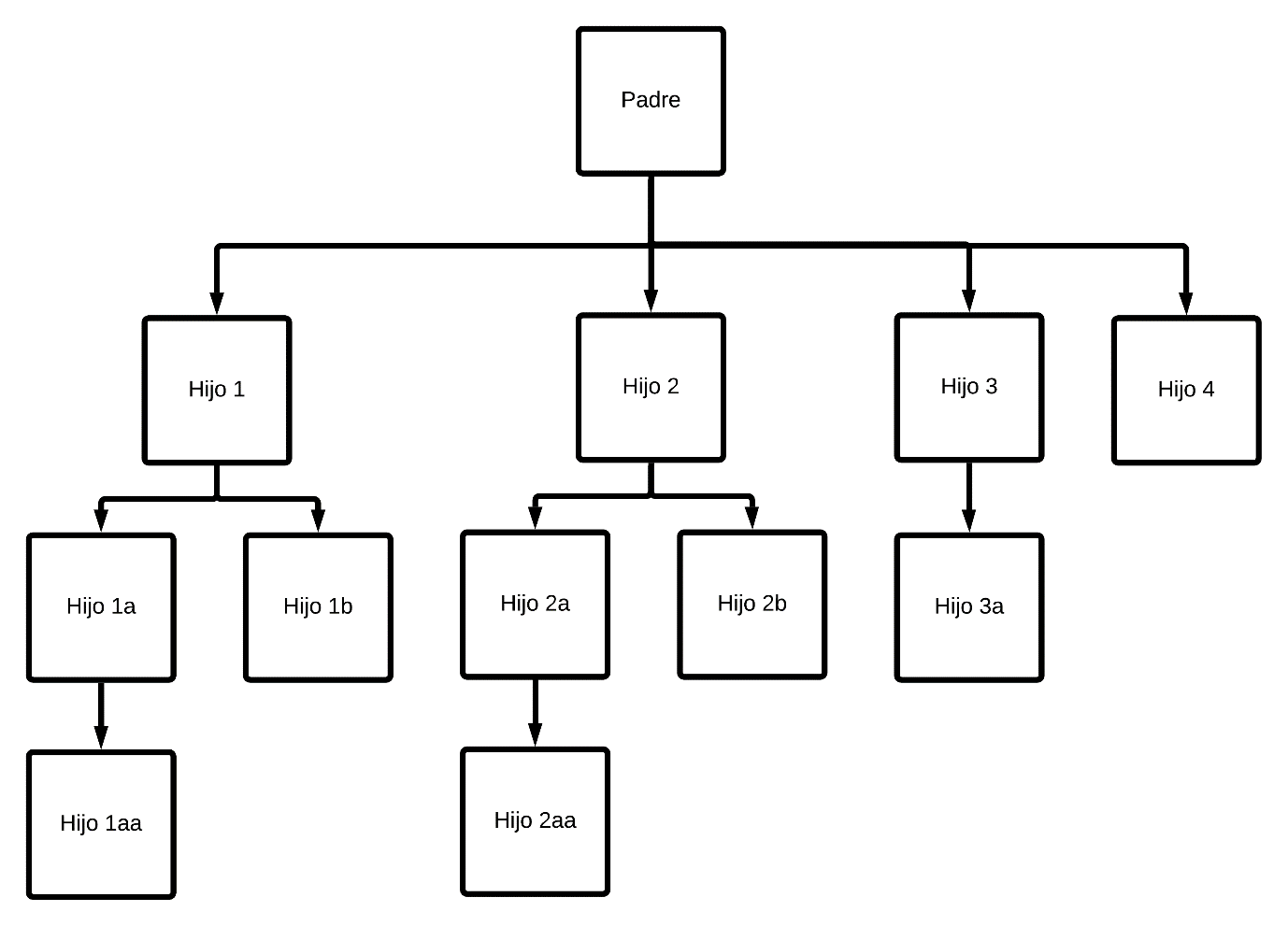
    }

    nuevo = fork();

    nuevo = fork();

    printf("Soy el proceso %d y mi padre es %d\n", getpid(), getppid());

}



4. Considerando el siguiente fragmento de código:

    for (num = 1; num <= n; num++){

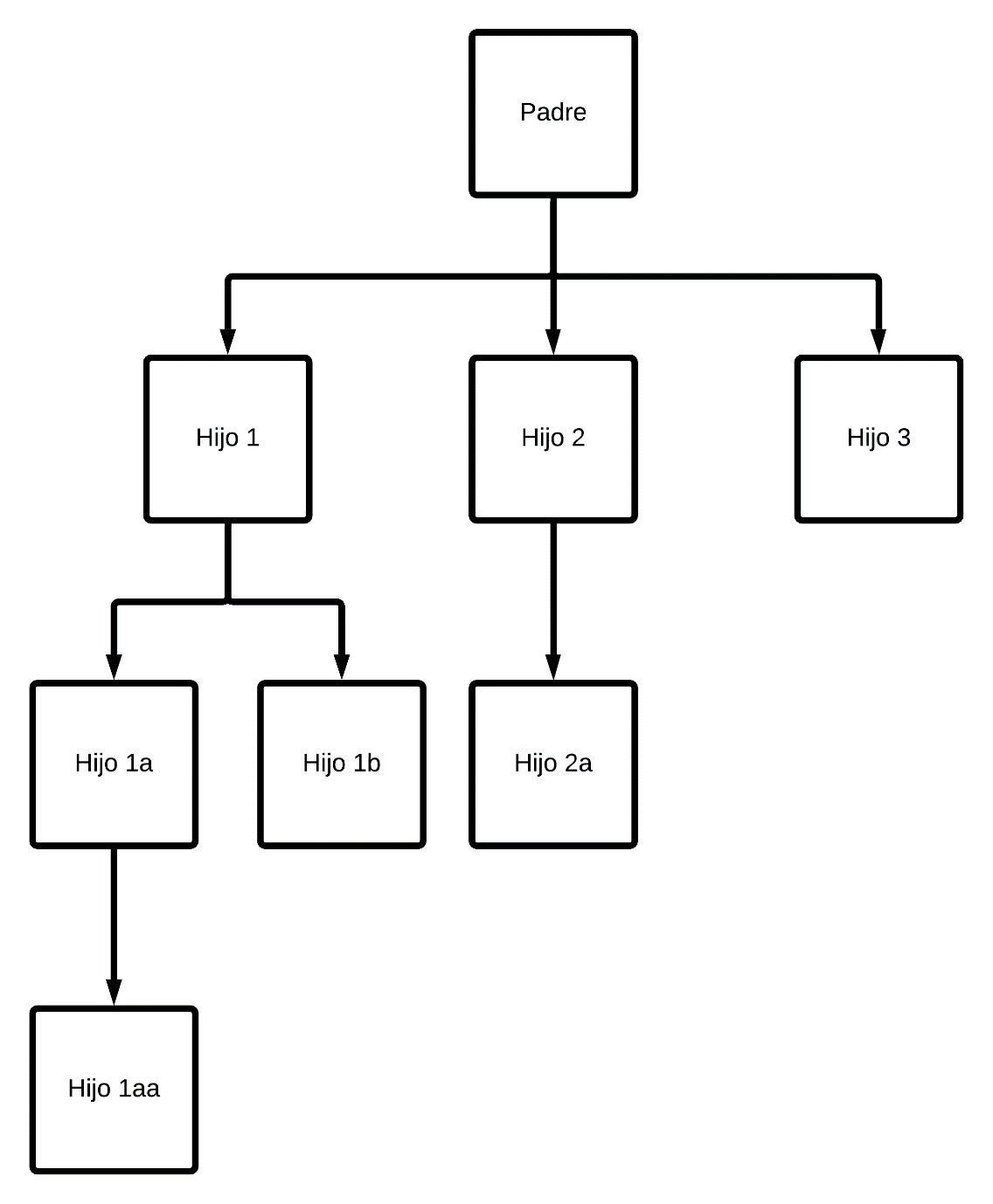
        nuevo = fork();

        if ((num == n) && (nuevo == 0))

            execlp ("ls", "ls", "-l", NULL);

    }

1. **Dibuja la jerarquía de procesos generada cuando se ejecuta y *n* es 3.**



1. **Indica en qué procesos se ha cambiado la imagen del proceso usando la función *execlp*.**

En los procesos del hijo1aa, hijo1b, hijo2a y el hijo3

5. Dibuja la jerarquía de procesos que resulta de la ejecución del siguiente código.

Indica para cada nuevo proceso el valor de las variables *i* y *j* en el momento

de su creación.

for (i = 0; i < 2; i++) {

    pid = getpid();

    for (j = 0; j < i+2; j++) {

        nuevo = fork();

        if (nuevo != 0) {

            nuevo = fork();

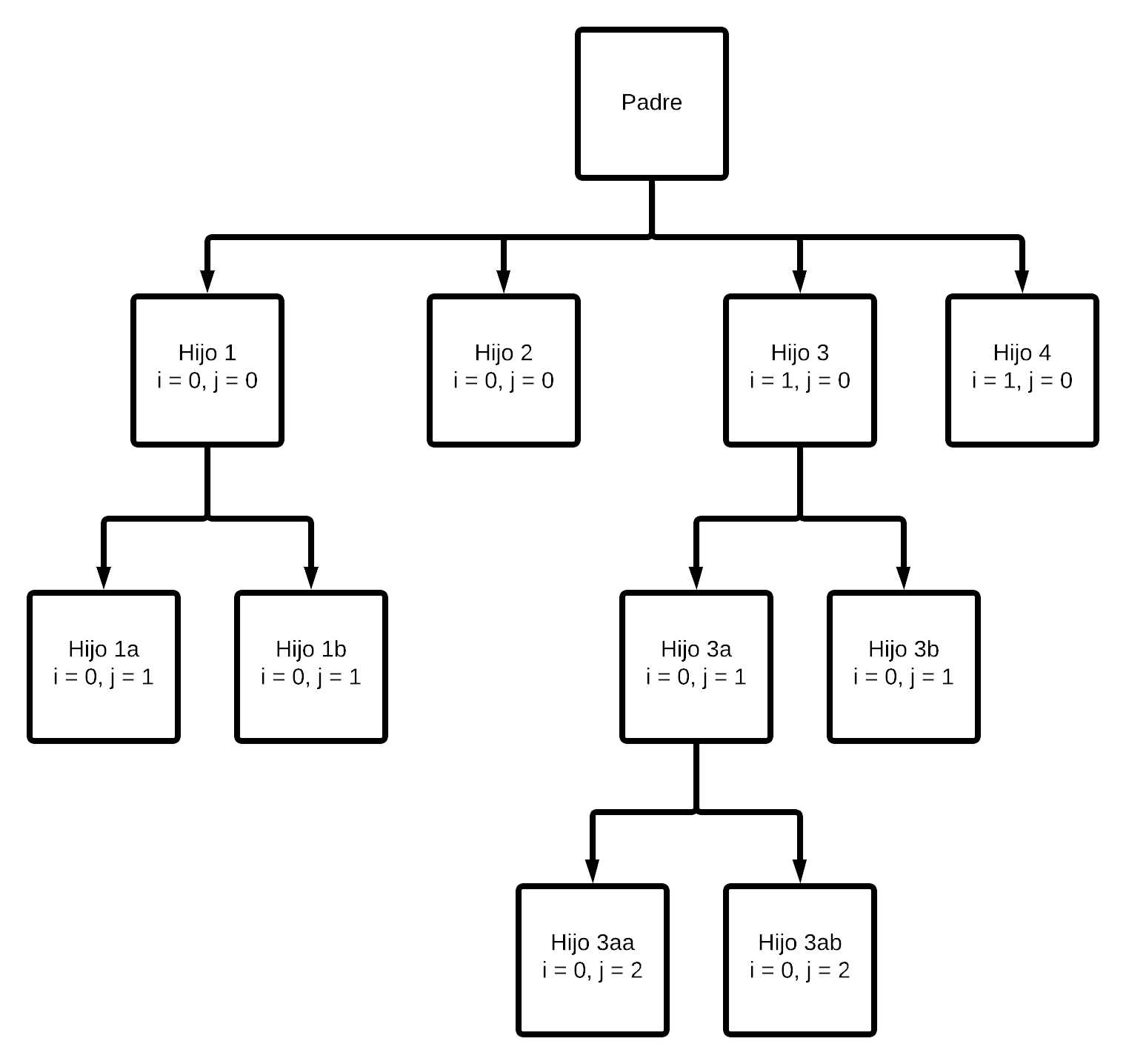
            break;

        }

    }

    if (pid != getpid()) break;

}



6. Estudia el siguiente código y escribe la jerarquía de procesos resultante. Después, compila y ejecuta el código para comprobarlo (deberás añadir llamadas al sistema *getpid*, *getppid* y *wait* para conseguirlo).

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#define L1 2

#define L2 3

int main (int argc, char ∗argv[]) {

    int cont1, cont2;

    pid t pid;

    for (cont2 = 0; cont2 < L2; cont2++) {

        for (cont1 = 0; cont1 < L1; cont1++) {

            pid = fork();

            if (pid == 0)    break;

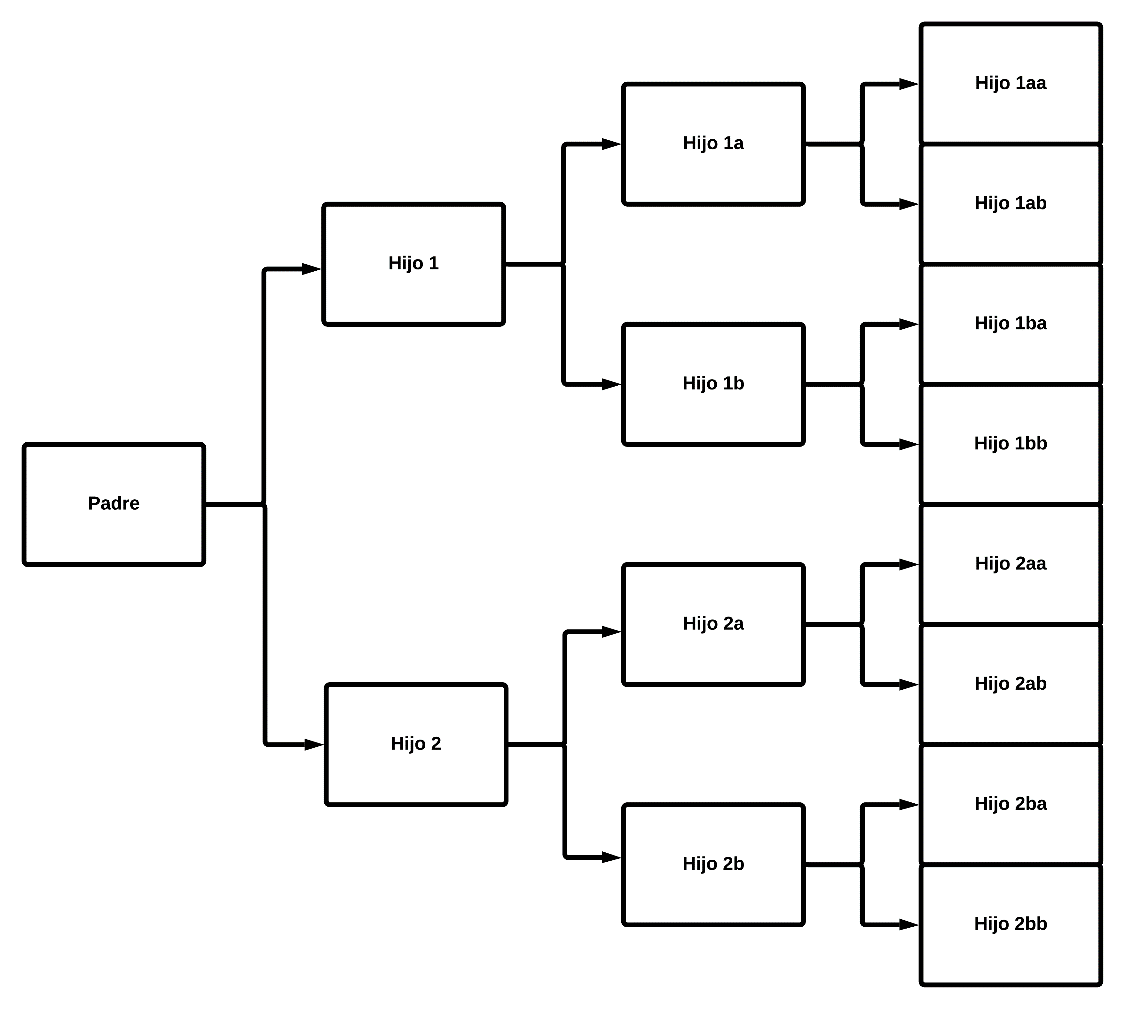
        }

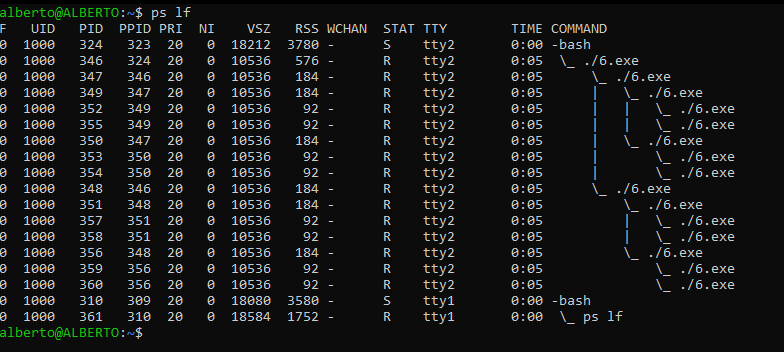
        if (pid!= 0)    break;

    }

    return 0;

}





Codigo utilizado

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#define L1 2

#define L2 3

int main (int argc, char argv[]) {

    int cont1, cont2;

    pid\_t pid;

    for (cont2 = 0; cont2 < L2; cont2++) {

        printf("\nSoy el proceso de PID %d y mi padre tiene %d de PID.", getpid(), getppid());

        for (cont1 = 0; cont1 < L1; cont1++) {

            pid = fork();

            if (pid == 0) break;

        }

        if (pid != 0) break;

    }

    if (pid == 0) {

        printf("\nFin del proceso de PID %d. \n", getpid());

        srand(getppid());

        sleep(rand()%5);

    }

    else printf("\nFin del proceso de PID %d. \n", wait(NULL));

    return 0;

}

