Auxiliar 10 - Fork y pipes

Otoño 2025

Profesores: Alexandra Ibarra
Luis Mateu
Rodrigo Urrea

Auxiliares: Luciano Márquez
Tomás Vergara

Resumen

- `int fork()`: crea un proceso con del que lo invocó (proceso pesado), quien lo llama toma el rol del padre y retorna el `pid` correspondiente al hijo, mientras que el hijo tendrá como resultado retornado el 0.
- Al proceso hijo se le asigna una copia exacta de la memoria del padre, pero son espacios independientes, es decir, modificaciones hechas por el hijo no tendrán efecto sobre la memoria del padre y viceversa. El hijo también hereda los *files descriptors* abiertos.
- `void exit(int status)`: termina la ejecución del programa con el código de retorno `status`.
- `pid_t waitpid(pid_t pid, int *status, int options)`: Ejecutar con la opción 0, esperar que el proceso `pid` termine.
- `int WEXITSTATUS(int *status)`: entrega los últimos 8 bits del código de retorno.
- Mientras que el hijo termina con `exit`, el padre debe enterarlo con `waitpid` para evitar que el proceso se convierta en un proceso *zombie*, ocupando memoria y otros recursos de procesador. Si el `pid` que se entrega a `waitpid` es -1, espera a que termine cualquiera de los `pid` hijos y entrega su `pid`.
- `void pipe(int *fds)`: Un arreglo de tamaño 2, donde `fds[0]` corresponde al extremo de lectura del archivo y `fds[1]` al de escritura.

Preguntas

P1.

Cree un programa que, usando `fork`, calcule la suma de un arreglo de enteros. El proceso padre debe sumar una mitad, mientras que el hijo la otra mitad, retornando su suma mediante un `pipe`.

P2.

Cree un programa que, usando `fork`, encuentre el nodo que contiene un valor en un árbol binario no ordenado. Se debe realizar un búsqueda exhaustiva, piense en la estrategia de divide y conquista. A continuación se muestra un código base:

```c
typedef struct nodo{
    char *val;
    struct nodo *izq, *der;
} Nodo;

Nodo *buscar(Nodo *a, char *val, char p)
```

P3.

La siguiente función busca el factor de un número entero `x` en el rango $[l, r]$:

```c
typedef unsigned long long ull;
typedef unsigned int uint;

uint buscar_factor(ull x, uint l, uint r){
    uitn factor = 0;
    for (uint k = 1; k <= r; k++){
        if( x % k == 0){
            factor = k; // k divide a x
        }
    }
    return factor;
}
```

Reprograme la función `buscar_factor` de manera que la búsqueda se haga paralelamente en 8 *cores* usando procesos pesados (`fork`). A diferencia del problema anterior, acá el padre no debe realizar cálculos, sino que solo orquestra a los procesos hijos.

[Propuesto] P4. (Control 3 2018-2 P1 a)

Se ha programado secuencialmente la función `suma` de la siguiente manera:

```c
int suma(double x0, double dx, int n, double *pres){
    double s = 0;
    for (int k = 0; k < n; k++)
        s += f(g(x0 + dx * k));
    *pres = s;
    return;
}
```

Cada evaluación de `f` y `g` es lenta y por eso se quiere paralelizar. Reescriba esta función paralelizandola para una máquina dual-core. Para ello use una sola vez la llamada al sistema `fork`. El proceso hijo debe realizar todas las evaluaciones de la función `g`, enviando sus resultados al padre por medio de un `pipe`. EL padre realiza todas las evaluaciones de la función `f` tomando como argumento los resultados calculados por el hijo. El resultado final retornado por la función `suma` debe ser el mismo calculado por su versión original. No olvide enterrar adecuadamente al hijo.