Taller Fork()

Nicolas Perez1
Pontificia Universidad Javeriana, Bogotá D.C, Colombia
penicolas@javeriana.edu.co

INTRODUCCIÓN

Este informe describe la implementación de un programa en C que utiliza la función **fork()** para crear procesos y **pipes** para la comunicación entre ellos. El código realiza la suma de dos arreglos de enteros, los cuales contienen valores leídos desde archivos. Cada proceso hijo realiza un cálculo y envía el resultado al proceso padre a través de un canal de comunicación.

DESCRIPCIÓN DE FUNCIONAMIENTO

- **Lectura de archivos**: Se define la función leer_fichero, que toma el nombre de un archivo y carga los enteros en un arreglo dinámico. Cada archivo contiene una serie de números que se sumarán en los procesos hijos.

```c
void leer_fichero(char *nombre_archivo, int *arreglo, int n) {
    FILE *f = fopen(nombre_archivo, "r");
    if (f == NULL) {
        perror("Error al abrir el archivo");
        exit(1);
    }

    // Leer los números del archivo y almacenarlos en el arreglo
    for (int i = 0; i < n; i++) {
        fscanf(f, "%d", &arreglo[i]);
    }

    fclose(f);
}
```

Figura 1. Función Leer Fichero

La función recibe como parámetro un puntero a un arreglo de caracteres que representan el nombre del archivo a leer, el apuntador a un arreglo de enteros, y la cantidad de “elementos” que tiene el archivo.

Luego recorre cada palabra del archivo y lo almacena como elemento en el arreglo hasta llegar al final del archivo.

- **Cálculo de la suma**: La función calcular_suma recibe un arreglo y calcula la suma de sus elementos. Se utiliza un **pipe** para enviar el resultado a otro proceso. Esta función es utilizada por los procesos nieto y segundo hijo para calcular la suma de los dos archivos.

```
void calcular_suma(int *arreglo, int n, int pipefd[2], const char *proceso_nombre) {
    close(pipefd[0]);
    int suma = 0;

    //Se calcula la suma de todos los elementos del arreglo
    for (int i = 0; i < n; i++) {
        suma += arreglo[i];
    }

    //Se imprime el resultado de la suma con el nombre del proceso
    printf("%s, suma: %d\n", proceso_nombre, suma);

    //Se envía la suma por el pipe
    write(pipefd[1], &suma, sizeof(suma));
    close(pipefd[1]);
}
```

Figura 2. Función Calcular Suma

La función recibe un arreglo de enteros y recorre cada uno de sus elementos para sumarlos. En cada iteración, el valor del elemento actual del arreglo se añade a la variable suma, que acumula el resultado.

Una vez calculada la suma total, la función imprime el valor de la suma junto con el nombre del proceso que la ejecuta.

Después de imprimir la suma, se envía el valor de la suma a través del pipe, (pipefd[1]). El resultado puede ser leído por otro proceso, como el proceso padre o un hijo superior.

- **Creación de procesos con fork()**: El uso de **fork()** permite la creación de tres procesos:

```
int pipeA[2], pipeB[2], pipeC[2];
if (pipe(pipeA) == -1 || pipe(pipeB) == -1 || pipe(pipeC) == -1) {
    perror("Error al crear pipes");
    exit(1);
}
```

Figura 3. Creación de Pipes

- **Nieto**: Calcula la suma de los elementos del primer archivo y la envía a través de pipeA. (pID = 0_1)

```
} else if (pid_nieto == 0) {

    // El proceso nieto (grand hijo) calcula la suma del arreglo00 y la envía por pipeA
    calcular_suma(arreglo00, N1, pipeA, "Grand hijo");

    exit(0);
}
```

Figura 4. Proceso de Grand Hijo

- **Segundo hijo**: Calcula la suma del segundo archivo y la envía a través de pipeB. (pID = 0_2)

```
} else if (pid_segundo_hijo == 0) {
    // El proceso segundo hijo calcula la suma del arreglo01 y la envía por pipeB
    calcular_suma(arreglo01, N2, pipeB, "Segundo hijo");
    exit(0);
}
```

Figura 5. Proceso de Segundo Hijo

- **Primer hijo**: Recibe las sumas de ambos hijos y calcula la suma total, que es enviada al proceso padre por pipeC. (pID = 0_3)

```
else if (pid_primer_hijo == 0) {
  close(pipeC[0]);
  int suma_total = 0;
  int sumaA, sumaB;

  // Leer la suma del "Grand hijo" (arreglo00) desde el pipeA
  read(pipeA[0], &sumaA, sizeof(sumaA));

  // Leer la suma del "Segundo hijo" (arreglo01) desde el pipeB
  read(pipeB[0], &sumaB, sizeof(sumaB));

  // Sumar ambas sumas recibidas
  suma_total = sumaA + sumaB;

  // Imprimir el resultado de la suma total (para depuración)
  printf("Primer hijo, suma total: %d\n", suma_total);

  // Enviar la suma total al proceso padre usando el pipeC
  write(pipeC[1], &suma_total, sizeof(suma_total));
  close(pipeC[1]);
  exit(0);
```

Figura 6. Suma de sumas de proceso Primer Hijo

- **Comunicación con pipes**: Los **pipes** son utilizados para la comunicación entre los procesos. Cada hijo envía su resultado al proceso padre mediante un pipe, que recibe y gestiona la información necesaria.
    - **Resultado del Padre**:

```
// Proceso padre
// Se cierran los extremos de escritura de todos los pipes ya que el padre solo recibe datos
close(pipeA[1]); close(pipeB[1]); close(pipeC[1]);
// Se cierran los extremos de lectura de pipeA y pipeB, ya no los necesitamos
close(pipeA[0]); close(pipeB[0]);

int suma_total;

// Leer la suma total enviada por el primer hijo desde el pipeC
read(pipeC[0], &suma_total, sizeof(suma_total));

// Imprimir la suma total que el proceso padre ha recibido
printf("Padre, suma total: %d\n", suma_total);

// Esperar a que terminen los procesos hijos
for (int i = 0; i < 3; i++) {
    wait(NULL);
}

// Liberar memoria dinámica
free(arreglo00);
free(arreglo01);
```

Figura 7. Cierre de Comunicación e Impresión de resultado de proceso Padre

Se cierran ambos extremos (escritura y lectura de pipeA y pipeB) ya que no serán necesarios.

Lee el valor de la suma total que fue calculada y enviada por el primer hijo a través del pipeC, usando el extremo de lectura del pipe piceC[0], y se almacena en la variable suma_total. Luego, se imprime este resultado almacenado como parte del proceso del padre. Y se espera que los procesos terminen la ejecución que verifica el bucle para que no se finalice antes que los hijos.

Y se libera la memoria usada.

CONCLUSIONES

Con este taller se demuestra claramente cómo se puede utilizar **fork()** para crear varios procesos y cómo los **pipes** facilitan la transmisión de información entre ellos. La combinación de estas herramientas es muy buena para situaciones en las que se necesita dividir el trabajo en varias tareas que se ejecutan al mismo tiempo, como en este caso, donde distintos procesos realizan cálculos y luego se comparten los resultados. Este tipo de implementación es muy útil para proyectos que requieren manejar operaciones paralelas o múltiples tareas al mismo tiempo. Aplicar estas técnicas correctamente no solo asegura que el programa funcione de manera correcta, sino que también abre las puertas a soluciones más avanzadas en futuros desarrollos.