



MATRIZ DE PIPES



Mendoza Flores Erick Gabriel
Nava Vivas Ana Paola
2CM8

Introducción

Generalmente un programa de computadora requiere cierto uso del CPU. Para esto, el sistema operativo asigna un proceso a dicho programa y lo envía al CPU para ser ejecutado. A veces, hay programas que realizan trabajos demasiado complejos y necesitan demasiado tiempo de procesamiento para ser ejecutados. Afortunadamente, C nos ofrece la herramienta *fork* para delegar el trabajo de calcular ciertas partes del programa a diferentes procesos. Estos procesos a veces necesitan comunicarse con el proceso que los creó, para hacerle saber que ha terminado y a veces para hacerle llegar cierta información.

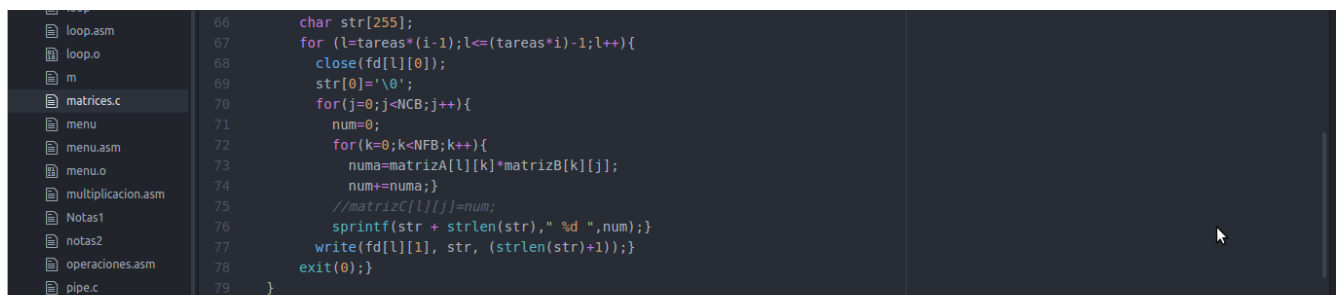
Objetivo

En esta práctica haremos uso de otra herramienta llamada *pipe* para poder establecer dicho canal de comunicación.

Desarrollo.

Se desarrollan dos programas distintos que resuelven el problema, así que se añadirán las capturas de ambos programas y enfoques. Los dos desarrollos hacen uso de los mismos algoritmos de asignación de procesos y del mismo procedimiento para el uso de *pipes*.

No modular :



```
66     char str[255];
67     for (l=tareas*(i-1);l<=(tareas*i)-1;l++){
68         close(fd[l][0]);
69         str[0]='\0';
70         for(j=0;j<NCB;j++){
71             num=0;
72             for(k=0;k<NFB;k++){
73                 numa=matrizA[l][k]*matrizB[k][j];
74                 num+=numa;}
75             //matrizC[l][j]=num;
76             sprintf(str + strlen(str)," %d ",num);}
77             write(fd[l][1], str, (strlen(str)+1));
78         exit(0);}
79     }
```

Modular :

```

void procCalc(int ini,int fin,
              int cA_rB,int colsB,
              int ** A,int ** B,int (*fd)[2])
{
    int it1,it2,it3,accum;
    char str[255];
    //printf("ini[%d]\n",ini);
    for(ini;ini<=fin;ini++)
    {
        close(fd[ini][0]);
        str[0]='\0';
        sprintf(str+strlen(str),"%d",ini);
        for(it1=0;it1<colsB;it1++)
        {
            for(it2=0,accum=0;it2<cA_rB;it2++) accum+=A[ini][it2]*B[it2][it1];
            sprintf(str+strlen(str),"\t[%d]\t",accum);
        }
        sprintf(str+strlen(str),"\n",ini);
        write(fd[ini][1],str,strlen(str)+1);
    }
    exit(0);
}

```

El programa recibe el tamaño de las matrices a multiplicar, cuidando siempre que ambas matrices sean multiplicables entre sí.

```

12 printf("\numero de filas matriz A: ");
13 scanf("%d",&NFA);
14 printf("\numero de columnas A= numero de filas B: ");
15 scanf("%d",&NCA);
16 NFB=NCA;
17 printf("\numero de columnas B: ");
18 scanf("%d",&NCB);
19 printf("\numero de procesos:\n");
20 scanf("%d",&nump);

```

```

void inputD(int*rowsA,int*cA_rB,int*colsB,int*procs)
{
    printf("\n# filas en A: "); scanf("%d",rowsA);
    printf("\n# columnas en A (filas en B): "); scanf("%d",cA_rB);
    printf("\n# columnas en B: "); scanf("%d",colsB);
    printf("\n# de procesos: "); scanf("%d",procs);
    if(*rowsA<=0||*cA_rB<=0||*colsB<=0||*procs<=0) puts("\nAlgún tamaño no es válido.\n");
    if(*rowsA<*procs){puts("\nNo pueden haber más procesos que filas.\nTerminando...");exit(0);}
}

```

Al recibir los datos, se crean matrices dentro del programa, se inicializan con los parámetros adquiridos y se llenan de forma dinámica de acuerdo a su tamaño con números aleatorios del 0 al 9 (1 al 9 en el caso del programa modular).

El programa recibe el número de procesos necesarios que el usuario requiera utilizar para realizar el cálculo y realiza el cálculo correspondiente para utilizar esa información en el algoritmo de selección de filas.

Se crea un arreglo de *pipes* correspondiente al número de filas de la matriz principal. Se abren los *pipes*.

```

39 int (*fd)[2];
40 fd=malloc(NFA*sizeof(int[2]));
41 for(i=0;i<NFA;i++){
42     pipe(fd[i]);
43 }

```

El programa genera los procesos hijos y ejecuta la función de multiplicación de matrices.

Se aplica el algoritmo y se realiza la multiplicación de las matrices de acuerdo a éste, seleccionando las filas correspondientes por proceso, y cada resultado “envía” al padre cada fila calculada. Independientemente de los procesos utilizados, el programa envía cada fila (con formato de cadena) por un *pipe* independiente.

Conclusiones

En la práctica, el uso de los *fork* están ya limitados, es importante lograr que tengan todas las funcionalidades posibles y que pueda existir una comunicación certera entre procesos. Existen también los *pipes* con nombre, que aumentan la funcionalidad entre los procesos hermanos o procesos no relacionados entre sí.

