Kevin Briones Fuentes

INFORME

Capturas de su código explicando su solución

```
int main(int argc, char *argv[])
{
  if(argc <2){
          printf("Por favor, debe ingresar dos argumentos \n 1- Indicador si va
 a ingresar una ruta del archivo. \n 2. La ruta del archivo o los números a ord
enar separados por coma. \n\n NOTA: Los números a ingresar deben ser separados
por coma. \n Por Ejemplo: \n 1 -    ./a.out -f archivo.txt \n 2 -    ./a.out -n 1
,4,6,2,20,15 \n\n");
          exit(1);
          return 1;
  }
  key_t k;
  int numbers_quantity = 0;
  int array_numbers[10000];
  int i;
  char *option;
  int id;
  int mat_id;
  int *shm_array;
  char delimito[] = ",";
  char texto[10000];
```

Al ejecutar el programa se llama al método main el cual debe tener como argumentos argc, y un puntero a argv[]; para que al llamar al programa a ejecutar reconozca que se le deben mandar parámetros que para este ejercicio son necesarios. Estos argumentos son:

Para indicar cuantos parámetros se mandan, y para reconocer si el usuario que ejecuta el programa mandará un archivo, o a su vez envía una cadena de números separados por coma.

Posterior a esto, se verifica si el usuario ha mandado al menos dos parámetros; que en caso de que no lo haga, se debe indicar al usuario cuales son las opciones disponibles para ejecutar el programa.

```
  char texto[10000];
  if(argc != 3){
          if(strcmp(argv[1],"-h") == 0 ){
          printf("Por favor, debe ingresar dos argumentos \n 1- Indicador si va
 a ingresar una ruta del archivo. \n 2. La ruta del archivo o los números a ord
enar separados por coma. \n\n NOTA: Los números a ingresar deben ser separados
por coma. \n Por Ejemplo: \n 1 -    ./a.out -f archivo.txt \n 2 -    ./a.out -n 1
,4,6,2,20,15 \n\n");
          exit(1);
          return 1;
  }else{
          printf("Por favor, debe ingresar dos argumentos \n 1- Indicador si va
 a ingresar una ruta del archivo. \n 2. La ruta del archivo o los números a ord
enar separados por coma. \n\n NOTA: Los números a ingresar deben ser separados
por coma. \n");
          exit(1);
          return 1;
          }
  }
```

Abstrayendo lo que debe producir el programa, se establece una condición; en el cual si los argumentos no son iguales a 3; este mostrará un mensaje al usuario. Esto se da porque se debe pasar un argumento de bandera, para reconocer si:

1- El usuario va a leer un archivo.
2- El usuario pasará como parámetros una cadena de enteros separados por coma
3- El usuario pedirá un mensaje de ayuda.

Hay que notar que en la primera imagen se establecen algunos valores.

El valor k, que se usará como una clave identificadora de la memoria compartida destinada para el programa; numbers_quantity, array_numbers, que son valores del array de enteros que vamos a necesitar dadas las entradas del usuario; i para iterar en array; el puntero a char option que es un indicador de en que opción nos encontramos; el puntero a shm_array, que será un array que podrá compartirse entre procesos para que los datos concuerden; delimito y texto; que son necesarios para poder guardar los datos que me dará el usuario y separarlos en enteros.

```
if(strcmp(argv[1],"-f")==0){
        printf("Se realizará el ordenamiento de un archivo.\n");
        FILE *archivo = fopen(argv[2], "r");
        if(archivo == NULL){
                printf("Error abriendo el archivo. \n");
                exit(1);
                return 1;
        }
        char c;
        int add = 0;
        while(!feof(archivo)){
                c= fgetc(archivo);
                if(c!='\0' || c!='\n'){
                        texto[add] = c;
                }
                add++;
        }
        fclose(archivo);

        }
else if(strcmp(argv[1],"-n")==0){
        strcpy(texto,argv[2]);

}
```

Luego se hace una comparación; para observar si el argv ->1 tiene banderas conocidas:

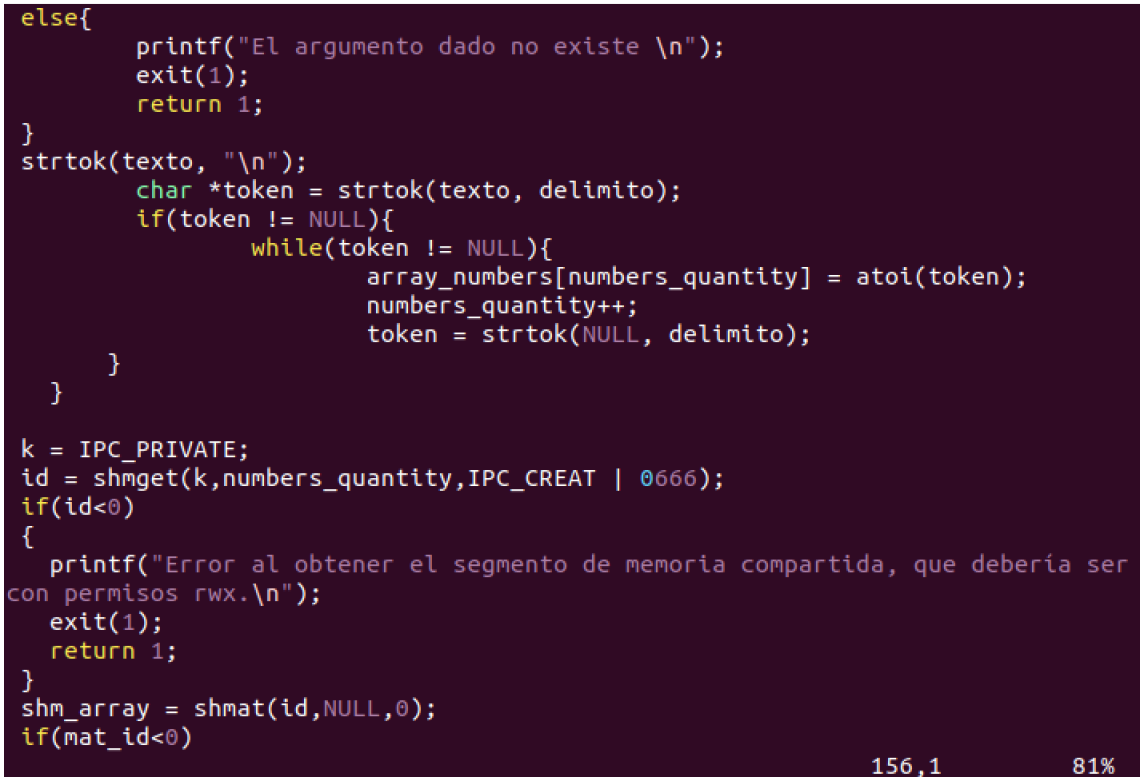
1- “-f” Para realizar ordenamiento de un archivo de texto.
2- “-n” Para realizar ordenamiento de una cadena de string, de enteros separados por coma.

Cuando se debe realizar una lectura de archivo de texto, se usa fopen para abrirlo con la bandera “r”, de modo que se abrirá en modo solo lectura.

Y en caso de que no se pueda abrir el archivo, se mostrará un mensaje de error al usuario.

En caso de que se realice la lectura del archivo, se leerá por cada carácter el archivo, añadiéndolo al puntero a char texto hasta que el puntero del archivo llegue al final; posterior se dará un cierre a la lectura del archivo.

Si fuera el caso de que se vaya a pasar por parámetros la cadena de string de enteros separados por coma, se copiará la información de argv ->1 al puntero a char texto.

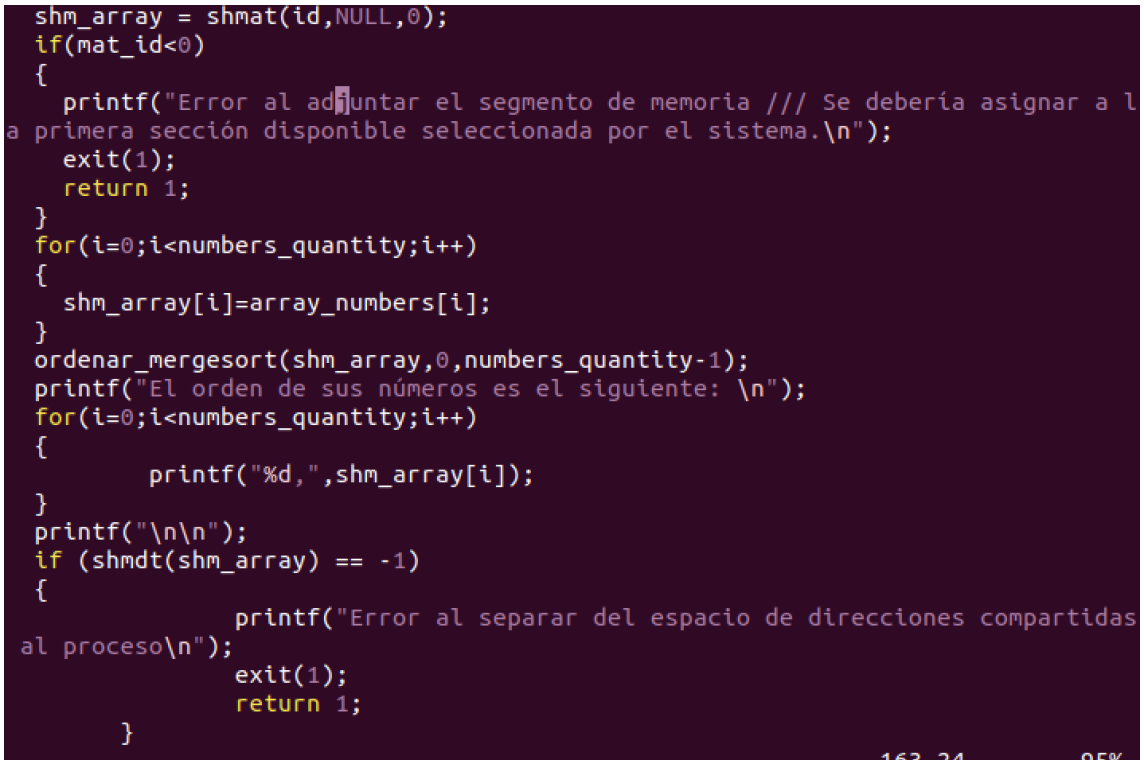


En caso de que los argumentos no sean conocidos, se manda un mensaje al usuario indicándolo.

Luego se quita los saltos de línea del texto que innecesariamente se podrían encontrar.

Ya teniendo los números puestos en un puntero a char; se los convierte a enteros con la función atoi, y se los guarda en array_numbers.

Luego de esto se llama a la función shmget, el cual obtendrá un segmento de memoria compartida libre que se pueda utilizar. En caso de que se pueda usar, se usará la función shmat para separar este segmento para nuestro uso en el shm_array.



```
  shm_array = shmat(id,NULL,0);
  if(mat_id<0)
  {
    printf("Error al adjuntar el segmento de memoria /// Se debería asignar a l
a primera sección disponible seleccionada por el sistema.\n");
    exit(1);
    return 1;
  }
  for(i=0;i<numbers_quantity;i++)
  {
    shm_array[i]=array_numbers[i];
  }
  ordenar_mergesort(shm_array,0,numbers_quantity-1);
  printf("El orden de sus números es el siguiente: \n");
  for(i=0;i<numbers_quantity;i++)
  {
          printf("%d,",shm_array[i]);
  }
  printf("\n\n");
  if (shmdt(shm_array) == -1)
  {
                printf("Error al separar del espacio de direcciones compartidas
 al proceso\n");
                exit(1);
                return 1;
        }
```

Al tener ya separado shm_array como un array que podrá ser compartido, se llama la función ordenar_mergesort, pasandole como argumentos el array, el lado izquierdo del array, y el lado derecho. Aquí el array será ordenado y posteriormente se imprimirá este array, abstrayendo por ahora lo que hace la función.

Para que el programa finalice, se separa del espacio de memoria compartida, y luego se quita el segmento.

Kevin Briones Fuentes

```
void ordenar_mergesort(int valores[],int left,int right)
{
  int medium=(left+right)/2;
  int estado;
  int lc,rc;
  if(left>=right)
  {
    return;
  }
  lc=fork();
  if(lc<0)
  {
    printf("Error al crear proceso hijo\n");
    exit(1);
  }
  else if(lc==0)
  {
    ordenar_mergesort(valores,left,medium);
    exit(0);
  }
  else
  {
    rc=fork();
    if(rc<0)
    {
      printf("Error al crear proceso hijo con la porcion derecha\n");
```

```
      exit(1);
    }
    else if(rc==0)
    {
      ordenar_mergesort(valores,medium+1,right);
      exit(0);
    }
  }
  waitpid(lc,&estado,0);
  waitpid(rc,&estado,0);
  ordenar(valores,left,medium,right);
}
```

Ahora teniendo ya el array de enteros que se tiene que ordenar, y una forma de comunicarse entre procesos, se projascede a realizar el algoritmo mergesort.

Por ello primero se define la mitad del array, y se realiza un fork para el lado izquierdo del array; y si al crear el proceso hijo se realiza con éxito, este proceso hijo procede nuevamente a contar la mitad del array; considerando el nuevo lado izquierdo y derecho. Cuando cada proceso intente seguir prosiguiendo con su ejecución, se le manda un waitpid, de modo que ningún proceso podrá avanzar hasta que se llegue a los últimos hijos (serán las hojas y siempre los que se encuentren del lado más izquierdo del array).

Cuando se llegue al último hijo, su ejecución prosigue con la función ordenar, al cual se le manda el array de valores a ordenar, la posición más izquierda del array, la más derecha y la mitad. Por ahora se abstrae lo que realiza.

Posterior a ello, si ya no se pueden realizar más procesos hijos en la parte izquierda, se hace fork con el lado derecho, y se procede nuevamente a llamar al mismo algoritmo ordenar_mergesort(…) para que llegue a la hoja izquierda del lado derecho. De modo que al final, todas las hojas son evaluadas y llamadas mediante procesos.

```
void ordenar(int valores[],int left,int medium,int right)
{
  int n1=medium-left+1,n2=right-medium,i,j,k;
  int l[n1],r[n2];
  for(i=0;i<n1;i++)
  {
    l[i]=valores[left+i];
  }
  for(i=0;i<n2;i++)
  {
    r[i]=valores[medium+i+1];
  }
  i=0;
  j=0;
  k=left;
  while(i<n1 && j<n2)
  {
    if(l[i]<r[j])
    {
      valores[k++]=l[i++];
    }
    else
    {
      valores[k++]=r[j++];
    }
  }
  while(i<n1)
  {
    valores[k++]=l[i++];
  }
  while (j<n2)
  {
    valores[k++]=r[j++];
  }
}
```

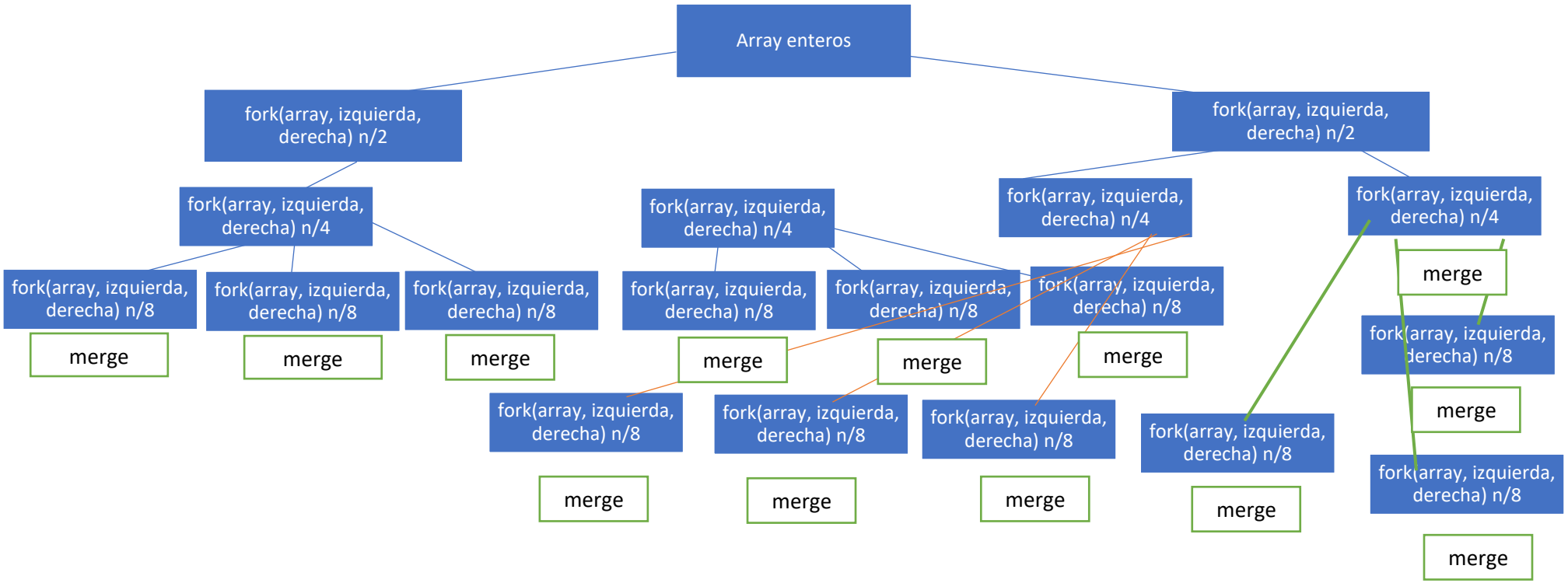
Ahora lo que realiza la función ordenar, es que cada cuando el algoritmo ordenar_mergesort lo requierda; ordena en el array valores, los valores desde cierta posición izquierda, hasta cierta posición derecha. Todo esto considerando la mitad de donde actualmente se está tomando el array. Y como mínimo izquierdo desde la posición médium-left+1, y como máximo izquierdo la posición right-medium.

Con ello se establece la cantidad de la porción izquierda que se comparará entre ellos, y posteriormente la cantidad de la porción derecha que se va a comparar.

Al final, dada la cantidad máxima izquierda, y la máxima derecha, se comparan ambas listas, de modo que el valor más mínimo entre las dos siempre se reemplazará y se colocará a la izquierda, mientras que el valor mayor siempre se pondrá a la derecha

Kevin Briones Fuentes

Diagrama de su solución



Bibliografía

GÓMEZ LABRADOR, R. A. M. Ó. N. M. (1996, junio). Comunicación entre Procesos UNIX. PROGRAMACIÓN DE SISTEMAS UNIX: COMUNICACIÓN ENTRE PROCESOS. http://es.tldp.org/Universitarios/seminario-1.html

Parallel Merge Sort — Parallel Sorting. (s. f.). selkie macalester. http://www.selkie.macalester.edu/csinparallel/modules/ParallelSorting/build/html/MergeSort/MergeSort.html

tutorialspoint. (s. f.). waitpid() - Unix, Linux System Call. https://www.tutorialspoint.com/unix_system_calls/waitpid.htm

IBM. (s. f.). shmdt() — Shared memory detach operation. IBM Documentation. https://www.ibm.com/docs/en/zos/2.4.0?topic=functions-shmdt-shared-memory-detach-operation

mkssoftware. (s. f.). shmat() -- attach shared memory. https://www.mkssoftware.com/docs/man3/shmat.3.asp

Jacqui López. (2019, 26 abril). Memoria Dinámica uso de malloc en C. [Vídeo]. YouTube. https://www.youtube.com/watch?v=VSbkcFOoWdI

makigas: tutoriales de programación. (2020, 3 noviembre). Tutorial de C – 3x02. Archivos (2): leer con fgetc [Vídeo]. YouTube. https://www.youtube.com/watch?v=HIL53hk-ApY