OPERATING SYSTEM

MID SEMESTER LAB EXAM

R SHREJA
COE18B043

QUESTION

Implement a multiprocessing version of the following: 2 processes that sort an original array into two sorted lists and a third merging processes that merges the two sorted lists to generate the final sorted array.

CODE USING PIPES:

```c
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#define RD_END 0

#define WR_END 1


//function to sort an array

void swap(int *xp, int *yp){

   int temp = *xp;

   *xp = *yp;

   *yp = temp;

}

 void sort(int arr[], int n) {

  for (int i=0;i< n-1;i++)

      for (int j= 0;j< n-i-1;j++)

          if (arr[j] > arr[j+1])

             swap(&arr[j], &arr[j+1]);

}


//function to merge two sorted array

void merge_two_sorted_arrays(int arr1[], int arr2[], int arr3[], int m, int n){

   int i,j,k;

   i = j = k = 0;

   for(i=0;i< m && j < n;){

       if(arr1[i] < arr2[j]){

           arr3[k] = arr1[i];
```

```
            k++;
            i++;
    }
        else{
            arr3[k] = arr2[j];
            k++;
            j++;
        }
    }

    while(i < m){
        arr3[k] = arr1[i];
        k++;
        i++;
    }

    while(j < n){
        arr3[k] = arr2[j];
        k++;
        j++;
    }
}


//Driver code
int main(int argc,char *argv[]){
    //init clock
    float start_t, end_t, total_t;


    //start the clock
    start_t = clock();


    //Validate the input
    if(argc!= atoi(argv[1])+2){
        fprintf(stderr,"Error:Valid syntax ./sort [sizeof(arr)] arr[0:size-1]");
        exit(0);
    }
    //Input variables declaration and instantiation
```

```
int n=atoi(argv[1]); //array size
int arr[n];//Input array


printf("The array to be sorted is:\n");
for (int i = 0; i < n; ++i){
    arr[i]= atoi(argv[i+2]);
    printf("%d ",arr[i]);
}
printf("\n");


int n1=n/2;//first array size
int arr1[n1];//first array init


for (int i = 0; i < n1; ++i)
    arr1[i]=arr[i];


int n2=n-n1;//second array size
int arr2[n2],p=0;//second array init
for(int i=n/2;i<n;i++){
    arr2[p]=arr[i];
    p++;
}


//Output variable declarations
int arr3[n];//Output array


//Create two pipes for parent and child read and write.
int p1fd[2],p2fd[2];


//creation and validation of pipe
if(pipe(p1fd)==-1){
    fprintf(stderr,"Error:Unable to create pipe p1\n");
    return 1;
}
if(pipe(p2fd)==-1){
    fprintf(stderr,"Error:Unable to create pipe p2\n");
    return 1;
```

```
}


//fork
int pid1=fork();
//Validation of the fork
if(pid1<0)
    fprintf(stderr,"Error:fork of process1 failed!!\n");



if(pid1==0){
    //close unwanted pipes
    close(p1fd[RD_END]);
    //sort the first array
    sort(arr1,n1);


    printf("\nThe first list sorted @ process1....\n");
    for(int i=0;i<n1;i++)
        printf("%d ",arr1[i]);
    printf("\n");
    //write into the pipe
    write(p1fd[WR_END],&arr1,sizeof(arr1));
    exit(0);


}
//create the second process
int pid2=fork();


if(pid2<0)
    fprintf(stderr,"Error:fork of process2 failed!!\n");


if(pid2==0){
    //close unwanted pipes
    close(p2fd[RD_END]);


    //sort the second array
    sort(arr2,n2);


    printf("\nThe second list sorted @ process2...\n");
```

```
    for(int i=0;i<n2;i++)
        printf("%d ",arr2[i]);
    printf("\n");
    //write into the pipe
    write(p2fd[WR_END],&arr2,sizeof(arr2));
    exit(0);

}

//third process
if(pid1){

    wait(NULL);
    //close unwanted pipes
    close(p1fd[WR_END]);
    close(p2fd[WR_END]);

    //create the input arrays to receive from the process 1 and 2
    int arr4[n1],arr5[n2];

    // read from pipe 1
    read(p1fd[RD_END],&arr4,sizeof(arr4));

    //print out what is read
    printf("\nThe first sorted list received @ process3: \n");
    for (int i = 0; i < n1; ++i){
        printf("%d ",arr4[i]);
    }

    //read from pipe 2
    read(p2fd[RD_END],&arr5,sizeof(arr5));

    //print out what is read
    printf("\nThe second sorted list received @ process3: \n");
    for (int i = 0; i < n2; ++i){
        printf("%d ",arr5[i]);
    }
```

```
    //merge the two sorted arrays into a new sorted array
    printf("\nMerging the obtained lists @ process3 \n");
    merge_two_sorted_arrays(arr4,arr5,arr3,n1,n2);

    //print out the  final sorted array
    for (int m = 0; m< n; ++m)
        printf("%d ",arr3[m]);
    printf("\n");

}

   printf("\n");

   //end the clock
   end_t = clock();

   //calculate runtime  and print it out
   total_t = (1.0)*(end_t - start_t) / CLOCKS_PER_SEC;
   printf("Total time taken by CPU: %f\n", total_t );
   printf("Exiting of the program...\n");

}
```

OUTPUT:

```
shreja@lostinspace:~/Desktop/OS_LAB_midsem$ ./mid_sem_para 30 2 1 1 3 4 6 7 8 9 10 1 34 2 1 33 5 67 8 9 78 0 90 100 2 3 45 6 9 7 8
The array to be sorted is:
2 1 1 3 4 6 7 8 9 10 1 34 2 1 33 5 67 8 9 78 0 90 100 2 3 45 6 9 7 8

The first list sorted @ process1....
1 1 1 1 2 2 3 4 6 7 8 9 10 33 34

The second list sorted @ process2...
0 2 3 5 6 7 8 8 9 9 45 67 78 90 100

The first sorted list received @ process3:
1 1 1 1 2 2 3 4 6 7 8 9 10 33 34
The second sorted list received @ process3:
0 2 3 5 6 7 8 8 9 9 45 67 78 90 100
Merging the obtained lists @ process3
0 1 1 1 1 2 2 2 3 3 4 5 6 6 7 7 8 8 8 9 9 9 10 33 34 45 67 78 90 100

Total time taken by CPU: 0.000429
Exiting of the program...
shreja@lostinspace:~/Desktop/OS_LAB_midsem$ ./mid_sem_para 30 2 1 1 3 4 6 7 8 9 10 1 34 2 1 33 5 67 8 9 78 0 90 100 2 40 2
Error:Valid syntax ./sort [sizeof(arr)] arr[0:size-1]
shreja@lostinspace:~/Desktop/OS_LAB_midsem$
```

EXPLANATION:

The code is implemented using IPC(pipes). Steps of implementation are:

1. The input is taken in the command line size and the array.
2. Then we separate the array into two halves first one having gif(n/2) elements and the second having n-gif(n/2).[where gif(.) is the greatest integer function]
3. Created 2 processes and sorted each half of the array in them using the sort function
4. Passed both the sorted arrays into the third process using pipes and merged both of them into the final sorted array.
5. Also timed the whole program run time.

As the input array is small and time is consumed in creating parent and children is higher than the process runtime itself, so the serial version given below is roughly 2 times faster than the multiprocessing version.I simulated the sort for 10000 input values and the parallel code was about 10 times faster than the serial code.

Description of functions used :

A.
```
void merge_two_sorted_arrays(int arr1[], int arr2[], int arr3[], int m, int n)
```

1. Create an array arr3[] of size n1 + n2.
2. Simultaneously traverse arr1[] and arr2[].
    - Pick smaller of current elements in arr1[] and arr2[], copy this smaller element to next position in arr3[] and move ahead in arr3[] and the array whose element is picked.
3. If there are remaining elements in arr1[] or arr2[], copy them also in arr3[].

```
void swap(int *xp, int *yp)
```

This is the trivial swap function used swap values and it swaps by reference. It is used in the sort function.

```
void sort(int arr[], int n)
```

This is the trivial bubble sort function.

CODE WITHOUT PIPES:

```c
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>



//function to sort an array

void swap(int *xp, int *yp){

   int temp = *xp;

   *xp = *yp;

   *yp = temp;

}

 void sort(int arr[], int n) {

  for (int i=0;i< n-1;i++)

      for (int j= 0;j< n-i-1;j++)

          if (arr[j] > arr[j+1])

             swap(&arr[j], &arr[j+1]);

}



//function to merge two sorted array

void merge_two_sorted_arrays(int arr1[], int arr2[], int arr3[], int m, int n){

   int i,j,k;

   i = j = k = 0;

   for(i=0;i< m && j < n;){

       if(arr1[i] < arr2[j]){

           arr3[k] = arr1[i];

            k++;

            i++;

   }

       else{

           arr3[k] = arr2[j];

           k++;

           j++;

       }
```

```
}


while(i < m){
    arr3[k] = arr1[i];
    k++;
    i++;
}


while(j < n){
    arr3[k] = arr2[j];
    k++;
    j++;
   }
}



//Driver code
int main(int argc,char *argv[]){
   //init clock
   float start_t, end_t, total_t;


   //start the clock
   start_t = clock();



   //Validate the input
   if(argc!= atoi(argv[1])+2){
    fprintf(stderr,"Error:Valid syntax ./sort [sizeof(arr)] arr[0:size-1]\n");
    exit(0);
}
//Input variables declaration and instantiation
int n=atoi(argv[1]); //array size
int arr[n];//Input array


printf("The array to be sorted is:\n");
for (int i = 0; i < n; ++i){
    arr[i]= atoi(argv[i+2]);
    printf("%d ",arr[i]);
```

```
}
printf("\n");


int n1=n/2;//first array size
int arr1[n1];//first array init


for (int i = 0; i < n1; ++i)
    arr1[i]=arr[i];


int n2=n-n1;//second array size
int arr2[n2],p=0;//second array init
for(int i=n/2;i<n;i++){
    arr2[p]=arr[i];
    p++;
}


//Output variable declarations
int arr3[n];//Output array


sort(arr1,n1);//sort first half array
    printf("\nThe first list sorted\n");
    //print out the sorted array
    for(int i=0;i<n1;i++)
        printf("%d ",arr1[i]);
    printf("\n");


    sort(arr2,n2);//sort second half of the array
    //print out the sorted array
    printf("\nThe second list sorted\n");
    for(int i=0;i<n2;i++)
        printf("%d ",arr2[i]);
    printf("\n");


    //merge the two sorted arry into the final sorted array
    printf("\nMerging the obtained lists  \n");
    merge_two_sorted_arrays(arr1,arr2,arr3,n1,n2);


    //printout the final sorted array
```

```c
    for (int m = 0; m< n; ++m)
        printf("%d ",arr3[m]);
    printf("\n");


printf("\n");


//end the clock
end_t = clock();


//calculate runtime  and print it out
total_t = (1.0)*(end_t - start_t) / CLOCKS_PER_SEC;
printf("Total time taken by CPU: %f\n", total_t );

  printf("Exiting of the program...\n");


}
```

OUTPUT:

```
shreja@lostinspace:~/Desktop/OS_LAB_midsem$ ./mid_sem_ser 30 2 1 1 3 4 6 7 8 9 10 1 34 2 1 33 5 67 8 9 78 0 90 100 2 3 45 6 9 7 8
The array to be sorted is:
2 1 1 3 4 6 7 8 9 10 1 34 2 1 33 5 67 8 9 78 0 90 100 2 3 45 6 9 7 8

The first list sorted
1 1 1 1 2 2 3 4 6 7 8 9 10 33 34

The second list sorted
0 2 3 5 6 7 8 8 9 9 45 67 78 90 100

Merging the obtained lists
0 1 1 1 1 2 2 2 3 3 4 5 6 6 7 7 8 8 8 9 9 9 10 33 34 45 67 78 90 100

Total time taken by CPU: 0.000197
Exiting of the program...
shreja@lostinspace:~/Desktop/OS_LAB_midsem$ ./mid_sem_ser 30 2 1 1 3 4 6 7 8 9 10 1 34 2 1 33 5 67 8 9 78 0 90 100 2 3 45 6 9 7
Error:Valid syntax ./sort [sizeof(arr)] arr[0:size-1]
```

EXPLANATION:

1. Serial is implemented the same as the multi-process version just without the pipes.
2. This program is to compare the runtimes of the multi-process with the serial implementation.

As seen above for small inputs the serial is 2 times faster than the multi-process code.