OSL

Assignment – 2A

Process Control System Calls

**Roll no**: 33245

**CODE**:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

// Bubble Sort

void bubbleSort(int arr[], int n)

{

int temp, i, j;

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

{

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

{

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

{

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

// Merge Sort

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

int main()

{

int n, i;

printf("Enter the number of integers you want to sort: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d integers:\n", n);

for (i = 0; i < n; i++)

{

scanf("%d", &arr[i]);

}

int choice;

printf("\nEnter your choice:\n");

printf("1. Fork, Wait, and Sort\n");

printf("2. For Orphan\n");

printf("3. For Zombie\n");

scanf("%d", &choice);

switch (choice)

{

case 1:

{

pid\_t pid = fork();

if (pid < 0)

{

printf("Fork failed.\n");

exit(1);

}

else if (pid == 0)

{

printf("\nChild process, Bubble Sort started.\n");

bubbleSort(arr, n);

printf("\nSorted array by the child process ,Bubble Sort:\n");

for (i = 0; i < n; i++)

printf("%d ", arr[i]);

printf("\n");

}

else

{

printf("\nParent process ,Merge Sort started.\n");

mergeSort(arr, 0, n - 1);

printf("\nSorted array by the parent process ,Merge Sort:\n");

for (i = 0; i < n; i++)

printf("%d ", arr[i]);

printf("\n");

wait(NULL);

}

break;

}

case 2:

{

pid\_t pid = fork();

if (pid < 0)

{

printf("Fork failed.\n");

exit(1);

}

else if (pid == 0)

{

// Orphan process

printf("\nChild process started.\n");

printf("Printing pid in child process (PID: %d)\n", getpid());

printf("Printing ppid in child process(PID: %d) \n", getppid());

printf("Parent process terminated before the child process.\n");

sleep(5);

printf("Printing new pid in child process (PID: %d)\n", getpid());

printf("Printing new ppid in child process(PID: %d) \n", getppid());

char command[100];

sprintf(command, "ps -elf | grep %d", getpid());

system(command);

printf("Child(Orphan) process completed.\n");

wait(NULL);

}

else

{

// Parent process

printf("\nParent process started.\n");

printf("Printing pid in parent process (PID: %d)\n", getpid());

printf("Printing ppid in parent process(PID: %d) \n", getppid());

printf("\nParent process (PID: %d) completed.\n", getpid());

}

break;

}

case 3:

{

pid\_t pid = fork();

if (pid < 0)

{

printf("Fork failed.\n");

exit(1);

}

else if (pid == 0)

{

// Child process

printf("\nChild process started.\n");

printf("\nPrinting pid in child process (PID: %d)\n", getpid());

printf("\nPrinting ppid in child process(PID: %d) \n", getppid());

}

else

{

// Parent process

printf("\nParent process started.\n");

printf("Parent process will sleep to create a Zombie.\n");

sleep(10);

char command[100];

sprintf(command, "ps -elf | grep %d", getpid());

system(command);

// The parent process will complete before calling wait.

printf("\nParent process (PID: %d) completed.\n", getpid());

wait(NULL);

}

break;

}

default:

printf("Invalid choice.\n");

break;

}

return 0;

}

**OUTPUT**



